Formalization of Gyrovector Spaces as Models of Hyperbolic Geometry and Special Relativity

Jelena Marković and Filip Marić

April 9, 2025

Abstract

In this paper, we present an Isabelle/HOL formalization of noncommutative and nonassociative algebraic structures known as *gyrogroups* and *gyrovector spaces*. These concepts were introduced by Abraham A. Ungar [1] and have deep connections to hyperbolic geometry and special relativity. Gyrovector spaces can be used to define models of hyperbolic geometry. Unlike other models, gyrovector spaces offer the advantage that all definitions exhibit remarkable syntactical similarities to standard Euclidean and Cartesian geometry (e.g., points on the line between a and b satisfy the parametric equation $a \oplus t \otimes (\ominus a \oplus b)$, for $t \in \mathbb{R}$, while the hyperbolic Pythagorean theorem is expressed as $a^2 \oplus b^2 = c^2$, where $\otimes$, $\oplus$, and $\ominus$ represent gyro operations).

We begin by formally defining gyrogroups and gyrovector spaces and proving their numerous properties. Next, we formalize Möbius and Einstein models of these abstract structures (formulated in the two-dimensional, complex plane), and then demonstrate that these are equivalent to the Poincaré and Klein-Beltrami models, satisfying Tarski's geometry axioms for hyperbolic geometry.

Contents

```
theory GyroGroup
  imports Main
begin

class gyrogroupoid =
  fixes gyrozero :: 'a (0_g)
  fixes gyroplus :: 'a ⇒ 'a ⇒ 'a (infixl ⊕ 100)
begin
definition gyroaut :: ('a ⇒ 'a) ⇒ bool where
  gyroaut f ⟷
      (∀ a b. f (a ⊕ b) = f a ⊕ f b) ∧
      bij f
end
```

class *gyrogroup* = *gyrogroupoid* +
  fixes *gyroinv* :: $'a \Rightarrow 'a$ ($\ominus$)
  fixes *gyr* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
  assumes *gyro-left-id* [*simp*]: $\bigwedge a.\ 0_g \oplus a = a$
  assumes *gyro-left-inv* [*simp*]: $\bigwedge a.\ \ominus a \oplus a = 0_g$
  assumes *gyro-left-assoc*: $\bigwedge a\ b\ z.\ a \oplus (b \oplus z) = (a \oplus b) \oplus (gyr\ a\ b\ z)$
  assumes *gyr-left-loop*: $\bigwedge a\ b.\ gyr\ a\ b = gyr\ (a \oplus b)\ b$
  assumes *gyr-gyroaut*: $\bigwedge a\ b.\ gyroaut\ (gyr\ a\ b)$
begin

definition *gyrominus* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\ominus_b$ *100*) **where**
  $a \ominus_b b = a \oplus (\ominus\ b)$

end

context *gyrogroup*
begin

lemma *gyr-distrib* [*simp*]:
  $gyr\ a\ b\ (x \oplus y) = gyr\ a\ b\ x \oplus gyr\ a\ b\ y$
  by (*metis gyroaut-def gyr-gyroaut*)

lemma *gyr-inj*:
  assumes $gyr\ a\ b\ x = gyr\ a\ b\ y$
  shows $x = y$
  using *assms*
  by (*metis UNIV-I bij-betw-iff-bijections gyr-gyroaut gyroaut-def*)

Def 2.7, (2.2)

definition *cogyroplus* (**infixr** $\oplus_c$ *100*) **where**
  $a \oplus_c b = a \oplus (gyr\ a\ (\ominus b)\ b)$

definition *cogyrominus* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\ominus_{cb}$ *100*) **where**
  $a \ominus_{cb} b = a \oplus_c (\ominus\ b)$

definition *cogyroinv* ($\ominus_c$) **where**
  $\ominus_c\ a = 0_g \ominus_{cb} a$

Thm 2.8, (1)

lemma *gyro-left-cancel*:
  assumes $a \oplus b = a \oplus c$
  shows $b = c$
  using *assms*

proof−
  from *assms*
  have $(\ominus a) \oplus (a \oplus b) = (\ominus a) \oplus (a \oplus c)$
    by *simp*

then have $(\ominus a \oplus a) \oplus gyr\ (\ominus a)\ a\ b = (\ominus a \oplus a) \oplus gyr\ (\ominus a)\ a\ c$
using *gyro-left-assoc*
by *simp*
then have $gyr\ (\ominus a)\ a\ b = gyr\ (\ominus a)\ a\ c$
by *simp*
then show $b = c$
using *gyr-inj*
by *blast*
qed

Thm 2.8, (2)

definition *gyro-is-left-id* **where**
gyro-is-left-id $z \longleftrightarrow (\forall\ x.\ z \oplus x = x)$

lemma *gyro-is-left-id-0* [*simp*]:
shows *gyro-is-left-id* 0_g
by (*simp add*: *gyro-is-left-id-def*)

lemma *gyr-id-1'*:
assumes *gyro-is-left-id z*
shows $gyr\ z\ a = id$
using *assms*
unfolding *gyro-is-left-id-def*
by (*metis eq-id-iff gyro-left-cancel gyro-left-assoc*)

lemma *gyr-id-1* [*simp*]:
shows $gyr\ 0_g\ a = id$
using *gyr-id-1'*[*of* 0_g]
by *simp*

Thm 2.8, (3)

definition *gyro-is-left-inv* **where**
gyro-is-left-inv $x\ a \longleftrightarrow x \oplus a = 0_g$

definition *gyro-is-right-inv* **where**
gyro-is-right-inv $x\ a \longleftrightarrow a \oplus x = 0_g$

lemma *gyro-is-left-inv* [*simp*]:
shows *gyro-is-left-inv* $(\ominus a)\ a$
by (*simp add*: *gyro-is-left-inv-def*)

lemma *gyr-inv-1'*:
assumes *gyro-is-left-inv x a*
shows $gyr\ x\ a = id$
using *assms gyr-left-loop*[*of x a*]
by (*simp add*: *gyro-is-left-inv-def*)

lemma *gyr-inv-1* [*simp*]:
shows $gyr\ (\ominus a)\ a = id$

using *gyr-left-loop*[*of* $\ominus a$ *a*]
by *simp*

Thm 2.8, (4)

lemma *gyr-id* [*simp*]:
shows *gyr a a = id*
by (*metis gyr-id-1 gyr-left-loop gyro-left-id*)

Thm 2.8, (5)

lemma *gyro-right-id* [*simp*]:
shows $a \oplus 0_g = a$
proof−
have $\ominus a \oplus (a \oplus 0_g) = \ominus a \oplus a$
using *gyro-left-assoc*
by *simp*
thus *?thesis*
using *gyro-left-cancel*[*of* $\ominus a$]
by *simp*
qed

lemma *gyro-inv-id* [*simp*]: $\ominus\ 0_g = 0_g$
by (*metis gyro-left-inv gyro-right-id*)

Thm 2.8, (6)

lemma *gyro-left-id-unique*:
assumes *gyro-is-left-id z*
shows $z = 0_g$
proof−
have $0_g = z \oplus 0_g$
using *assms*
by (*metis gyro-is-left-id-def*)
thus *?thesis*
using *gyro-right-id*[*of z*]
by *simp*
qed

Thm 2.8, (7)

lemma *gyro-left-inv-right-inv*:
assumes *gyro-is-left-inv x a*
shows *gyro-is-right-inv x a*
using *assms*
by (*metis gyr-inv-1′ gyro-left-cancel gyro-right-id id-apply gyro-is-left-inv-def gyro-is-right-inv-def gyro-left-assoc*)

lemma *gyro-rigth-inv* [*simp*]:
shows $a \oplus (\ominus a) = 0_g$
using *gyro-is-right-inv-def gyro-left-inv-right-inv*
by *simp*

Thm 2.8, (8)

lemma
  assumes *gyro-is-left-inv x a*
  shows $x = \ominus a$
  using *assms*
  by (*metis gyr-inv-1 id-apply gyro-is-left-inv-def gyro-left-assoc gyro-left-id gyro-left-inv*)

Thm 2.8, (9)

lemma *gyro-left-cancel'*:
  shows $\ominus\ a \oplus (a \oplus b) = b$
  by (*simp add*: *gyro-left-assoc*)

Thm 2.8, (10)

lemma *gyr-def*:
  shows $gyr\ a\ b\ x = \ominus\ (a \oplus b) \oplus (a \oplus (b \oplus x))$
  by (*metis gyro-left-cancel' gyro-left-assoc*)

Thm 2.8, (11)

lemma *gyr-id-3*:
  shows $gyr\ a\ b\ 0_g = 0_g$
  by (*simp add*: *gyr-def*)

Thm 2.8, (12)

lemma *gyr-inv-3*:
  shows $gyr\ a\ b\ (\ominus x) = \ominus\ (gyr\ a\ b\ x)$
  by (*metis gyroaut-def gyr-gyroaut gyr-id-3 gyro-left-cancel gyro-rigth-inv*)

Thm 2.8, (13)

lemma *gyr-id-2* [*simp*]:
  shows $gyr\ a\ 0_g = id$
  by (*metis gyro-left-cancel eq-id-iff gyro-left-assoc gyro-left-id gyro-right-id*)

lemma *gyr-distrib-gyrominus*:
  shows $gyr\ a\ b\ (c \ominus_b d) = gyr\ a\ b\ c \ominus_b gyr\ a\ b\ d$
  by (*metis gyroaut-def gyr-gyroaut gyr-inv-3 gyrominus-def*)

lemma *gyro-inv-idem* [*simp*]:
  shows $\ominus\ (\ominus\ a) = a$
  by (*metis gyr-inv-1 gyro-left-cancel gyro-left-assoc gyro-left-id gyro-left-inv*)

lemma *gyr-inv-2* [*simp*]:
  shows $gyr\ a\ (\ominus\ a) = id$
  using *gyr-inv-1*[*of* $\ominus a$]
  by *simp*

(2.3.a)

lemma *cogyro-left-id*:
  shows $0_g \oplus_c a = a$
  by (*simp add*: *cogyroplus-def gyr-id-3*)

(2.3.b)

lemma *cogyro-rigth-id*:
shows $a \oplus_c 0_g = a$
by (*simp add*: *cogyroplus-def gyr-id-3*)

(2.4)

lemma *cogyrominus*:
shows $a \ominus_{cb} b = a \ominus_b gyr\ a\ b\ b$
by (*simp add*: *cogyrominus-def cogyroplus-def gyr-inv-3 gyrominus-def*)

(2.7)

lemma *cogyro-right-inv*:
shows $a \oplus_c (\ominus_c\ a) = 0_g$
by (*metis cogyroinv-def cogyrominus-def cogyroplus-def gyr-inv-2 gyro-inv-idem gyro-right-id gyro-rigth-inv iso-tuple-update-accessor-eq-assist-idI gyr-left-loop gyro-left-assoc update-accessor-accessor-eqE*)

(2.6)

lemma *cogyro-left-inv*:
shows $(\ominus_c\ a) \oplus_c a = 0_g$
by (*metis cogyroinv-def cogyro-left-id cogyrominus-def cogyro-right-inv gyro-inv-idem*)

(2.8)

lemma *cogyro-gyro-inv*:
shows $\ominus_c\ a = \ominus\ a$
by (*simp add*: *cogyroinv-def cogyro-left-id cogyrominus-def*)

Thm 2.9, (2.9)

lemma *gyr-nested-1*:
shows $gyr\ a\ (b \oplus c) \circ gyr\ b\ c = gyr\ (a \oplus b)\ (gyr\ a\ b\ c) \circ gyr\ a\ b$ (**is** *?lhs* = *?rhs*)
proof
fix x

have $a \oplus (b \oplus (c \oplus x)) = (a \oplus b) \oplus gyr\ a\ b\ (c \oplus x)$
by (*simp add*: *gyro-left-assoc*[*of a b*])
also have ... $= (a \oplus b \oplus (gyr\ a\ b\ c \oplus gyr\ a\ b\ x))$
by *simp*
also have ... $= ((a \oplus b) \oplus gyr\ a\ b\ c) \oplus gyr\ (a \oplus b)\ (gyr\ a\ b\ c)\ (gyr\ a\ b\ x)$
by (*simp add*: *gyro-left-assoc*)
also have ... $= (a \oplus (b \oplus c)) \oplus gyr\ (a \oplus b)\ (gyr\ a\ b\ c)\ (gyr\ a\ b\ x)$
by (*simp add*: *gyro-left-assoc*)
finally
have *1*: $a \oplus (b \oplus (c \oplus x)) = (a \oplus (b \oplus c)) \oplus gyr\ (a \oplus b)\ (gyr\ a\ b\ c)\ (gyr\ a\ b\ x)$
.

have $a \oplus (b \oplus (c \oplus x)) = a \oplus (b \oplus c \oplus gyr\ b\ c\ x)$
by (*simp add*: *gyro-left-assoc*)

also have ... = $(a \oplus (b \oplus c)) \oplus$ *gyr* a $(b \oplus c)$ (*gyr* b c x)
  by (*simp add*: *gyro-left-assoc*)
finally have *2*: $a \oplus (b \oplus (c \oplus x)) = (a \oplus (b \oplus c)) \oplus$ *gyr* a $(b \oplus c)$ (*gyr* b c x)
  .

have *gyr* $(a \oplus b)$ (*gyr* a b c) (*gyr* a b x) = *gyr* a $(b \oplus c)$ (*gyr* b c x)
  using *1 2*
  by (*metis gyro-left-cancel'*)

thus *?lhs* x = *?rhs* x
  by *simp*
qed

Thm 2.9, (2.15)

lemma *gyr-nested-1'*:
  shows *gyr* $(a \oplus b)$ ($\ominus$ (*gyr* a b b)) $\circ$ *gyr* a b = *id*
  by (*metis comp-id gyr-inv-2 gyr-inv-3 gyr-nested-1 gyr-id-2 gyro-rigth-inv*)

Thm 2.9, (2.10)

lemma *gyr-nested-2*:
  shows *gyr* a ($\ominus$ (*gyr* a b b)) $\circ$ *gyr* a b = *id*
proof−
  have *gyr* $(a \oplus b)$ (*gyr* a b $(\ominus b)$) = *gyr* a ($\ominus$ (*gyr* a b b))
    by (*metis gyr-inv-3 gyro-left-assoc gyr-left-loop gyro-right-id gyro-rigth-inv*)
  thus *?thesis*
    using *gyr-nested-1*[*of* a b $\ominus$ b]
    by *simp*
qed

Thm 2.9, (2.11)

lemma *gyr-auto-id1*:
  shows *gyr* $(\ominus a)$ $(a \oplus b)$ $\circ$ *gyr* a b = *id*
  using *gyr-nested-1*[*of* $\ominus$ a a b]
  by *simp*

Thm 2.9, (2.12)

lemma *gyr-auto-id2*:
  shows *gyr* b $(a \oplus b)$ $\circ$ *gyr* a b = *id*
  by (*metis gyr-auto-id1 gyro-left-cancel' gyr-left-loop*)

Thm 2.10, (2.18)

lemma *gyro-plus-def-co*:
  shows $a \oplus b = a \oplus_c$ *gyr* a b b
  by (*simp add*: *cogyroplus-def gyr-nested-2 pointfree-idE*)

Thm 2.11, (2.21)

lemma *gyro-polygonal-addition-lemma*:
  shows $(\ominus a \oplus b) \oplus$ *gyr* $(\ominus a)$ b $(\ominus b \oplus c) = \ominus a \oplus c$
proof−

have $gyr\ (\ominus a)\ b\ (\ominus\ b \oplus c) = gyr\ (\ominus a)\ b\ (\ominus\ b) \oplus gyr\ (\ominus\ a)\ b\ c$
by *simp*
hence $(\ominus\ a \oplus b) \oplus gyr\ (\ominus a)\ b\ (\ominus\ b \oplus c) =$
$(\ominus\ a \oplus b) \oplus (gyr\ (\ominus a)\ b\ (\ominus\ b) \oplus gyr\ (\ominus\ a)\ b\ c)$
by *simp*
also have ... $= ((\ominus\ a \oplus b) \ominus_b gyr\ (\ominus\ a)\ b\ b) \oplus (gyr\ ((\ominus\ a) \oplus b)\ (\ominus\ (gyr\ (\ominus\ a)\ b\ b)) \circ gyr\ (\ominus\ a)\ b)\ c$
by (*metis calculation gyr-inv-3 gyr-nested-1′ gyro-left-cancel′ gyrominus-def gyro-right-id id-apply gyro-left-assoc*)
also have ... $= (\ominus\ a \oplus (b \ominus_b b)) \oplus c$
by (*metis gyr-inv-3 gyr-nested-1′ gyrominus-def id-apply gyro-left-assoc*)
also have ... $= \ominus\ a \oplus c$
by (*simp add*: *gyrominus-def*)
finally
show *?thesis*
.
qed

Thm 2.12, (2.23)

lemma *gyro-translation-1*:
shows $\ominus\ (\ominus a \oplus b) \oplus (\ominus a \oplus c) = gyr\ (\ominus\ a)\ b\ (\ominus b \oplus c)$
by (*metis gyr-def gyro-left-cancel′*)

Thm 3.13, (3.33a)

lemma *gyro-translation-2a*:
shows $\ominus\ (a \oplus b) \oplus (a \oplus c) = gyr\ a\ b\ (\ominus b \oplus c)$
by (*metis gyr-def gyro-left-cancel′*)

definition *gyro-polygonal-add* $(\oplus_p)$ **where**
$\oplus_p\ a\ b\ c = (\ominus\ a \oplus b) \oplus gyr\ (\ominus\ a)\ b\ (\ominus\ b \oplus c)$

Thm 2.15, (2.34, 2.35)

lemma *gyro-equation-right*:
shows $a \oplus x = b \longleftrightarrow x = \ominus a \oplus b$
by (*metis gyro-left-cancel′*)

Thm 2.15, (2.36, 2.37)

lemma *gyro-equation-left*:
shows $x \oplus a = b \longleftrightarrow x = b \ominus_{cb} a$
by (*metis cogyrominus gyr-def gyr-inv-3 gyro-equation-right gyrominus-def gyro-right-id gyr-left-loop*)

lemma *oplus-ominus-cancel* [*simp*]:
shows $y = x \oplus (\ominus\ x \oplus y)$
by (*metis local.gyro-equation-right*)

(2.39)

lemma *cogyro-right-cancel′*:
shows $(b \ominus_{cb} a) \oplus a = b$

by (*simp add*: *gyro-equation-left*)

(2.40)

lemma *gyro-right-cancel′-dual*:
shows $(b \ominus_b a) \oplus_c a = b$
by (*metis cogyrominus-def gyro-equation-left gyro-inv-idem gyrominus-def*)

Thm 2.19 (2.48)

lemma *gyroaut-gyr-commute-lemma*:
  assumes *gyroaut A*
  shows $A \circ gyr\ a\ b = gyr\ (A\ a)\ (A\ b) \circ A$ (**is** *?lhs* = *?rhs*)
proof
  fix x
  have $(A\ a \oplus A\ b) \oplus (A \circ gyr\ a\ b)\ x = A((a \oplus b) \oplus gyr\ a\ b\ x)$
    using *assms gyroaut-def*
    by *auto*
  also have ... $= A\ (a \oplus (b \oplus x))$
    by (*simp add*: *gyro-left-assoc*)
  also have ... $= A\ a \oplus (A\ b \oplus A\ x)$
    using *assms gyroaut-def*
    by *auto*
  also have ... $= (A\ a \oplus A\ b) \oplus (gyr\ (A\ a)\ (A\ b)\ (A\ x))$
    by (*simp add*: *gyro-left-assoc*)
  finally
  show *?lhs* x = *?rhs* x
    using *gyro-left-cancel*
    by *auto*
qed

Thm 2.20

lemma *gyroaut-gyr-commute*:
  assumes *gyroaut A*
  shows $gyr\ a\ b = gyr\ (A\ a)\ (A\ b) \longleftrightarrow A \circ gyr\ a\ b = gyr\ a\ b \circ A$
proof
  assume $gyr\ a\ b = gyr\ (A\ a)\ (A\ b)$
  thus $A \circ gyr\ a\ b = gyr\ a\ b \circ A$
    using *gyroaut-gyr-commute-lemma*[*OF assms*]
    by *metis*
next
  assume ∗: $A \circ gyr\ a\ b = gyr\ a\ b \circ A$
  have $gyr\ (A\ a)\ (A\ b) = A \circ gyr\ a\ b \circ (inv\ A)$
    using *gyroaut-gyr-commute-lemma*[*OF assms*, *of a b*]
    by (*metis gyroaut-def assms bij-is-surj comp-id o-assoc surj-iff*)
  also have ... $= gyr\ a\ b$
    using ∗
    by (*metis gyroaut-def assms bij-is-surj comp-assoc comp-id surj-iff*)
  finally
  show $gyr\ a\ b = gyr\ (A\ a)\ (A\ b)$
    by *simp*

qed

2.50

lemma *gyr-commute-misc-1*:
shows $gyr\ (gyr\ a\ b\ a)\ (gyr\ a\ b\ b) = gyr\ a\ b$
by (*metis gyroaut-gyr-commute gyr-gyroaut*)

Thm 2.21 (2.52)

definition
$cogyroaut\ f \longleftrightarrow ((\forall\ a\ b.\ f\ (a \oplus_c b) = f\ a \oplus_c f\ b) \wedge bij\ f)$

lemma *gyro-coaut-iff-gyro-aut*:
shows $gyroaut\ f \longleftrightarrow cogyroaut\ f$
proof
assume *gyroaut f*
thus *cogyroaut f*
unfolding *gyroaut-def cogyroaut-def*
by (*smt cogyroplus-def gyr-def gyro-left-cancel gyro-right-id gyro-rigth-inv*)
next
assume *cogyroaut f*
thus *gyroaut f*
unfolding *gyroaut-def cogyroaut-def*
by (*metis bij-pointE cogyro-left-id cogyrominus-def cogyro-rigth-id gyro-equation-left gyro-right-cancel′-dual gyro-left-id gyrominus-def*)
qed

Thm 2.25, (2.76)

lemma *gyroplus-inv*:
shows $\ominus\ (a \oplus b) = gyr\ a\ b\ (\ominus\ b \ominus_b a)$
by (*metis gyr-def gyro-equation-right gyrominus-def gyro-rigth-inv*)

Thm 2.25, (2.77)

lemma *inv-gyr*:
shows $inv\ (gyr\ a\ b) = gyr\ (\ominus b)\ (\ominus a)$
by (*metis fun.map-id0 gyr-auto-id2 gyr-inv-2 gyr-nested-1 gyro-left-cancel′ gyrominus-def gyroplus-inv id-apply inv-unique-comp gyr-left-loop*)

Thm 2.26, (2.86)

lemma *gyr-aut-inv-1*:
shows $inv\ (gyr\ a\ b) = gyr\ a\ (\ominus\ (gyr\ a\ b\ b))$
by (*metis comp-eq-dest-lhs eq-id-iff gyr-nested-2 inv-unique-comp*)

Thm 2.26, (2.87)

lemma *gyr-aut-inv-2*:
shows $inv\ (gyr\ a\ b) = gyr\ (\ominus\ a)\ (a \oplus b)$
by (*metis gyr-auto-id2 gyro-left-cancel′ inv-unique-comp gyr-left-loop*)

Thm 2.26, (2.88)

lemma *gyr-aut-inv-3*:

shows *inv* (*gyr a b*) = *gyr b* ($a \oplus b$)
by (*metis gyr-auto-id2 gyro-left-cancel′ inv-unique-comp gyr-left-loop*)

Thm 2.26, (2.89)

lemma *gyr-1*:
shows *gyr a b* = *gyr b* ($\ominus b \ominus_b a$)
by (*metis inv-gyr gyr-aut-inv-2 gyro-inv-idem gyrominus-def*)

Thm 2.26, (2.90)

lemma *gyr-2*:
shows *gyr a b* = *gyr* ($\ominus a$) ($\ominus b \ominus_b a$)
using *inv-gyr gyr-aut-inv-3 gyrominus-def* **by** *auto*

Thm 2.26, (2.91)

lemma *gyr-3*:
shows *gyr a b* = *gyr* ($\ominus$ ($a \oplus b$)) *a*
by (*metis gyr-1 gyro-inv-idem gyro-left-cancel′ gyrominus-def*)

Thm 2.27, (2.92)

lemma *gyr-even*:
shows *gyr* ($\ominus$ *a*) ($\ominus$ *b*) = *gyr a b*
by (*metis cogyro-right-cancel′ gyr-aut-inv-3 inv-gyr local.gyr-left-loop*)

Thm 2.27, (2.93)

lemma *inv-gyr-sym*:
shows *inv* (*gyr a b*) = *gyr b a*
by (*simp add*: *inv-gyr gyr-even*)

Thm 2.27, (2.94a)

lemma *gyr-nested-3*:
shows *gyr b* ($\ominus$ (*gyr b a a*)) = *gyr a b*
using *gyr-aut-inv-1 inv-gyr-sym*
by *auto*

Thm 2.27, (2.94b)

lemma *gyr-nested-4*:
shows *gyr b* (*gyr b* ($\ominus$ *a*) *a*) = *gyr a* ($\ominus$ *b*)
by (*metis gyr-aut-inv-1 inv-gyr-sym gyr-even gyro-inv-idem*)

Thm 2.27, (2.94c)

lemma *gyr-nested-5*:
shows *gyr* ($\ominus$ (*gyr a b b*)) *a* = *gyr a b*
by (*metis inv-gyr-sym gyr-nested-3*)

Thm 2.27, (2.94d)

lemma *gyr-nested-6*:
shows *gyr* (*gyr a* ($\ominus$ *b*) *b*) *a* = *gyr a* ($\ominus$ *b*)
by (*metis inv-gyr inv-gyr-sym gyr-nested-4 gyro-inv-idem*)

Thm 2.28, (i)

lemma *gyro-right-assoc*:
shows $(a \oplus b) \oplus c = a \oplus (b \oplus gyr\ b\ a\ c)$
by (*metis gyr-aut-inv-2 inv-gyr-sym gyro-equation-right gyro-left-assoc*)

Thm 2.28, (ii)

lemma *gyr-right-loop*:
shows $gyr\ a\ b = gyr\ a\ (b \oplus a)$
using *gyr-aut-inv-3 inv-gyr-sym* **by** *auto*

Thm 2.29, (a)

lemma *gyr-left-coloop*:
shows $gyr\ a\ b = gyr\ (a \ominus_{cb}\ b)\ b$
by (*metis cogyro-right-cancel′ gyr-left-loop*)

Thm 2.29, (b)

lemma *gyr-rigth-coloop*:
shows $gyr\ a\ b = gyr\ a\ (b \ominus_{cb}\ a)$
by (*metis inv-gyr-sym gyr-left-coloop*)

Thm 2.30, (2.101a)

lemma *gyr-misc-1*:
shows $gyr\ (a \oplus b)\ (\ominus\ a) = gyr\ a\ b$
by (*metis gyr-aut-inv-2 inv-gyr-sym*)

Thm 2.30, (2.101b)

lemma *gyr-misc-2*:
shows $gyr\ (\ominus\ a)\ (a \oplus b) = gyr\ b\ a$
using *gyr-aut-inv-2 inv-gyr-sym* **by** *auto*

Thm 2.31, (2.103)

lemma *coautomorphic-inverse*:
shows $\ominus\ (a \oplus_c b) = (\ominus\ b) \oplus_c (\ominus\ a)$
proof−
have $a \oplus_c b = a \oplus gyr\ a\ (\ominus\ b)\ b$
by (*simp add*: *cogyroplus-def*)
also have ... $= \ominus\ (gyr\ a\ (gyr\ a\ (\ominus\ b)\ b)\ (\ominus\ (gyr\ a\ (\ominus\ b)\ b) \ominus_b a))$
by (*metis gyro-inv-idem gyroplus-inv*)
also have ... $= gyr\ a\ (\ominus\ (gyr\ a\ (\ominus\ b)\ (\ominus b)))\ (\ominus\ (\ominus\ (gyr\ a\ (\ominus\ b)\ b) \ominus_b a))$
by (*simp add*: *gyr-inv-3*)
also have ... $= (inv\ (gyr\ a\ (\ominus\ b)))\ (\ominus\ (\ominus\ (gyr\ a\ (\ominus\ b)\ b) \ominus_b a))$
by (*simp add*: *gyr-aut-inv-1*)
also have ... $= \ominus\ (\ominus\ b \ominus_b (inv\ (gyr\ a\ (\ominus\ b)))\ a)$
by (*metis cogyrominus cogyroplus-def inv-gyr-sym gyr-even gyr-inv-3 gyr-nested-3 gyro-equation-left gyro-inv-idem gyro-left-cancel′ gyrominus-def cogyro-right-cancel′ gyroplus-inv*)
also have ... $= \ominus\ ((\ominus\ b) \oplus_c (\ominus\ a))$
by (*simp add*: *cogyroplus-def inv-gyr-sym gyr-inv-3 gyrominus-def*)

finally
have $a \oplus_c b = \ominus ((\ominus b) \oplus_c (\ominus a))$
.
thus *?thesis*
by *simp*
qed

Thm 2.32, (2.105a)

lemma *gyr-misc-3*:
shows $gyr\ a\ b\ b = \ominus (\ominus (a \oplus b) \oplus a)$
by (*metis gyr-3 gyro-inv-idem gyro-left-cancel′ gyrominus-def gyroplus-inv*)

Thm 2.32, (2.105b)

lemma *gyr-misc-4*:
shows $gyr\ a\ (\ominus b)\ b = \ominus (a \ominus_b b) \oplus a$
by (*simp add*: *gyr-def gyrominus-def*)

Thm 2.35, (2.124)

lemma *mixed-gyroassoc-law*: $(a \oplus_c b) \oplus c = a \oplus gyr\ a\ (\ominus b)\ (b \oplus c)$
by (*metis* (*full-types*) *cogyroplus-def gyr-nested-6 gyro-right-assoc gyr-distrib*)

Thm 3.2

lemma *gyrocommute-iff-gyroatomorphic-inverse*:
shows $(\forall\ a\ b.\ \ominus (a \oplus b) = \ominus a \ominus_b b) \longleftrightarrow (\forall\ a\ b.\ a \oplus b = gyr\ a\ b\ (b \oplus a))$
by (*metis gyr-even gyro-inv-idem gyrominus-def gyroplus-inv*)

Thm 3.4

lemma *cogyro-commute-iff-gyrocommute*:
$(\forall\ a\ b.\ a \oplus_c b = b \oplus_c a) \longleftrightarrow (\forall\ a\ b.\ a \oplus b = gyr\ a\ b\ (b \oplus a))$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
have $\forall\ a\ b.\ a \oplus_c b = b \oplus_c a \longleftrightarrow \ominus (\ominus b \ominus_b gyr\ b\ (\ominus a)\ a) = b \oplus gyr\ b\ (\ominus a)\ a$
by (*metis coautomorphic-inverse cogyroplus-def gyr-even gyr-inv-3 gyro-inv-idem gyrominus-def*)
thus *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *cogyroplus-def gyr-even gyro-equation-right gyro-inv-idem gyrominus-def gyro-right-cancel′-dual gyroplus-inv*)
qed

end

class *gyrocommutative-gyrogroup* = *gyrogroup* +
assumes *gyro-commute*: $a \oplus b = gyr\ a\ b\ (b \oplus a)$
begin
lemma *gyroautomorphic-inverse*:
shows $\ominus (a \oplus b) = \ominus a \ominus_b b$
using *gyro-commute gyrocommute-iff-gyroatomorphic-inverse*
by *blast*

lemma *cogyro-commute*:

shows $a \oplus_c b = b \oplus_c a$
using *cogyro-commute-iff-gyrocommute gyro-commute*
by *blast*

Thm 3.5 (3.15)

lemma *gyr-commute-misc-2*:
shows $gyr\ a\ b \circ gyr\ (b \oplus a)\ c = gyr\ a\ (b \oplus c) \circ gyr\ b\ c$
by (*metis gyr-gyroaut gyroaut-gyr-commute-lemma gyr-nested-1 gyro-commute*)

Thm 3.6 (3.17, 3.18)

lemma *gyr-parallelogram*:
assumes $d = (b \oplus_c c) \ominus_b a$
shows $gyr\ a\ (\ominus\ b) \circ gyr\ b\ (\ominus\ c) \circ gyr\ c\ (\ominus\ d) = gyr\ a\ (\ominus\ d)$
proof−
have $*$: $\forall\ a'\ b'\ c'.\ gyr\ a'\ (b' \oplus a') \circ gyr\ (b' \oplus a')\ c' = gyr\ a'\ (b' \oplus c') \circ gyr\ (b' \oplus c')\ c'$
using *gyr-commute-misc-2 gyr-left-loop gyr-right-loop*
by *auto*
let $?a' = \ominus\ c$
let $?c' = \ominus\ a$
let $?b' = b \ominus_{cb} ?a'$

have $?b' \oplus ?c' = d$
by (*simp add*: *assms cogyrominus-def gyrominus-def*)
moreover
have $b \ominus_{cb} \ominus\ c \oplus \ominus\ c = b$
by (*simp add*: *cogyro-right-cancel'*)
ultimately
have $gyr\ (\ominus\ c)\ b \circ gyr\ b\ (\ominus\ a) = gyr\ (\ominus\ c)\ d \circ gyr\ d\ (\ominus\ a)$
using $*$[*rule-format*, *of ?a' ?b' ?c'*]
by *simp*
then show *?thesis*
by (*smt bij-is-inj gyroaut-def gyr-gyroaut inv-gyr-sym gyr-even gyro-inv-idem o-inv-distrib o-inv-o-cancel*)
qed

Thm 3.8 (3.23, 3.24)

lemma *gyr-parallelogram-iff*:
$d = (b \oplus_c c) \ominus_b a \longleftrightarrow \ominus c \oplus d = gyr\ c\ (\ominus b)\ (b \ominus_b a)$
proof−
have $(b \oplus_c c) \ominus_b a = (c \oplus_c b) \ominus_b (gyr\ b\ (\ominus c)\ (gyr\ c\ (\ominus b)\ a))$
by (*metis cogyro-commute local.gyr-def local.gyr-even local.gyro-right-assoc local.oplus-ominus-cancel*)

moreover have $(c \oplus_c b) \ominus_b (gyr\ b\ (\ominus c)\ (gyr\ c\ (\ominus b)\ a)) =$
$(c \oplus_c b) \ominus_b (gyr\ c\ (gyr\ c\ (\ominus b)\ b)\ (gyr\ c\ (\ominus b)\ a))$
using *local.gyr-nested-4* **by** *auto*

moreover have $(c \oplus_c b) \ominus_b (gyr\ c\ (gyr\ c\ (\ominus b)\ b)\ (gyr\ c\ (\ominus b)\ a)) =$

$c \oplus (gyr\ c\ (\ominus b)\ b) \ominus_b (gyr\ c\ (gyr\ c\ (\ominus b)\ b)\ (gyr\ c\ (\ominus b)\ a))$
using *local.cogyroplus-def* **by** *presburger*

moreover have $c \oplus (gyr\ c\ (\ominus b)\ b) \ominus_b (gyr\ c\ (gyr\ c\ (\ominus b)\ b)\ (gyr\ c\ (\ominus b)\ a)) =$
$c \oplus ((gyr\ c\ (\ominus b)\ b) \ominus_b (gyr\ c\ (\ominus b)\ a))$
by (*simp add*: *local.gyr-inv-3 local.gyro-left-assoc local.gyrominus-def*)
moreover have $(b \oplus_c c) \ominus_b a = c \oplus gyr\ c\ (\ominus b)\ (b \ominus_b a)$
by (*simp add*: *calculation*(*1*) *calculation*(*2*) *calculation*(*3*) *calculation*(*4*) *local.gyr-distrib-gyrominus*)
ultimately show *?thesis*
by (*metis local.oplus-ominus-cancel*)
qed

Thm 3.9 (3.26)

lemma *gyr-commute-misc-3*:
$gyr\ a\ b\ (b \oplus (a \oplus c)) = (a \oplus b) \oplus c$
using *gyr-distrib gyro-commute gyro-left-assoc gyro-right-assoc*
by (*metis* (*no-types*, *lifting*))

Thm 3.10 (3.28)

lemma *gyro-left-right-cancel*:
shows $(a \oplus b) \ominus_b a = gyr\ a\ b\ b$
by (*metis gyroautomorphic-inverse gyr-misc-3 gyro-inv-idem*)

Thm 3.11 (3.29)

lemma *cogyro-plus-def*:
shows $a \oplus_c b = a \oplus ((\ominus a \oplus b) \oplus a)$
by (*metis cogyro-commute-iff-gyrocommute cogyroplus-def gyro-commute gyro-equation-right gyro-right-assoc*)

Thm 3.12 (3.31)

lemma *cogyro-commute-misc1*:
shows $a \oplus_c (a \oplus b) = a \oplus (b \oplus a)$
by (*simp add*: *cogyro-plus-def gyro-left-cancel′*)

Thm 3.13 (3.33b)

lemma *gyro-translation-2b*:
shows $(a \oplus b) \ominus_b (a \oplus c) = gyr\ a\ b\ (b \ominus_b c)$
by (*metis gyr-commute-misc-3 gyroautomorphic-inverse gyro-equation-right gyrominus-def*)

Thm 3.14 (3.34)

(3.37)

lemma *gyr-commute-misc-4′*:
shows $gyr\ a\ (b \oplus c) = gyr\ a\ b \circ gyr\ (b \oplus a)\ c \circ gyr\ c\ b$
proof−
have $gyr\ a\ b \circ gyr\ (b \oplus a)\ c = gyr\ (a \oplus b)\ (gyr\ a\ b\ c) \circ gyr\ a\ b$

by (*simp add*: *gyr-commute-misc-2 local.gyr-nested-1*)
hence *gyr a* $(b \oplus c) \circ$ *gyr b c* = *gyr a b* $\circ$ *gyr* $(b \oplus a)$ *c*
by (*simp add*: *gyr-commute-misc-2*)
thus *?thesis*
by (*metis comp-assoc comp-id local.gyr-auto-id2 local.gyr-right-loop*)
qed

(3.38)

lemma *gyr-commute-misc-4''*:
shows *gyr* $(\ominus b \oplus d)$ $(b \oplus c)$ = *gyr* $(\ominus b)$ *d* $\circ$ *gyr d c* $\circ$ *gyr c b*
by (*metis gyr-commute-misc-4' local.gyr-misc-1 local.gyro-inv-idem local.gyro-left-cancel'*)

Thm 3.14 (3.34)

lemma *gyro-commute-misc-4*:
shows *gyr* $(\ominus a \oplus b)$ $(a \ominus_b c)$ = *gyr a* $(\ominus b) \circ$ *gyr b* $(\ominus c) \circ$ *gyr c* $(\ominus a)$
by (*metis gyr-commute-misc-4' gyr-even gyr-misc-1 gyro-inv-idem gyro-left-cancel' gyrominus-def*)

Thm 3.15 (3.40)

lemma *gyr-inv-2'*:
shows *gyr a* $(\ominus b)$ = *gyr* $(\ominus a \oplus b)$ $(a \oplus b) \circ$ *gyr a b*
by (*metis comp-id gyr-commute-misc-2 local.gyr-even local.gyr-id local.gyr-misc-1 local.gyro-inv-idem local.gyro-left-cancel'*)

Thm 3.17 (3.48)

lemma *gyr-master'*:
shows *gyr a x* $\circ$ *gyr* $(\ominus (x \oplus a))$ $(x \oplus b) \circ$ *gyr x b* = *gyr* $(\ominus a)$ *b*
by (*metis gyr-commute-misc-4' gyroautomorphic-inverse gyr-even gyr-misc-1 gyro-left-cancel' gyrominus-def*)

(3.51)

lemma *gyr-master*:
shows *gyr a x* $\circ$ *gyr* $(x \oplus a)$ $(\ominus (x \oplus b)) \circ$ *gyr x b* = *gyr* $(\ominus a)$ *b*
by (*metis gyr-master' gyr-even gyro-inv-idem*)

(3.52a)

lemma *gyr-master-misc1'*:
shows *gyr* $(\ominus a)$ *b* = *gyr* $(\ominus (a \oplus a))$ $(a \oplus b) \circ$ *gyr a b*
by (*metis fun.map-id gyr-master' local.gyr-id*)

(3.52b)

lemma *gyr-master-misc1''*:
shows *gyr* $(\ominus a)$ *b* = *gyr a b* $\circ$ *gyr* $(b \oplus a)$ $(\ominus (b \oplus b))$
by (*metis comp-id gyr-master gyr-id*)

(3.53a)

lemma *gyr-master-misc2'*:
shows *gyr* $(\ominus a \oplus b)$ $(a \oplus b)$ = *gyr* $(\ominus a)$ *b* $\circ$ *gyr b a*
by (*simp add*: *gyr-commute-misc-4''*)

(3.53b)

lemma *gyr-master-misc2''*:
  shows $gyr\ (\ominus a \oplus b)\ (a \oplus b) = gyr\ (\ominus\ a \oplus b)\ b \circ gyr\ b\ (a \oplus b)$
  using *gyr-master-misc2' local.gyr-left-loop local.gyr-right-loop*
  by *auto*

Thm 3.18 (3.60)

lemma $gyr\ a\ x \circ gyr\ (\ominus\ (gyr\ x\ a\ (a \ominus_b b)))\ (x \oplus b) \circ gyr\ x\ b = gyr\ a\ (\ominus\ b)$
  by (*metis gyr-master gyro-translation-2b gyr-even gyr-left-loop gyro-inv-idem gyrominus-def*)

definition *gyro-covariant* :: $nat \Rightarrow ('a\ list \Rightarrow 'a) \Rightarrow bool$ **where**
  $gyro\text{-}covariant\ n\ T \longleftrightarrow (\forall\ \tau\ xs.\ length\ xs = n \wedge gyroaut\ \tau \longrightarrow (\tau\ (T\ xs)) = T\ (map\ \tau\ xs)) \wedge$
  $(\forall\ x\ xs.\ length\ xs = n \longrightarrow x \oplus T\ xs = T\ (map\ (\lambda\ a.\ x \oplus a)\ xs))$

definition *gyro-covariant-3* :: $('a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a) \Rightarrow bool$ **where**
  $gyro\text{-}covariant\text{-}3\ T \longleftrightarrow (\forall\ \tau\ a\ b\ c.\ gyroaut\ \tau \longrightarrow (\tau\ (T\ a\ b\ c)) = T\ (\tau\ a)\ (\tau\ b)\ (\tau\ c)) \wedge$
  $(\forall\ x\ a\ b\ c.\ x \oplus T\ a\ b\ c = T\ (x \oplus a)\ (x \oplus b)\ (x \oplus c))$

lemma *gyro-covariant-3*:
  shows $gyro\text{-}covariant\text{-}3\ T \longleftrightarrow gyro\text{-}covariant\ 3\ (\lambda\ xs.\ T\ (xs\ !\ 0)\ (xs\ !\ 1)\ (xs\ !\ 2))$
  unfolding *gyro-covariant-3-def gyro-covariant-def*
  apply *safe*
    apply *simp*
   apply *simp*
  apply (*erule-tac* $x{=}\tau$ **in** *allE*, *erule-tac* $x{=}[a,\ b,\ c]$ **in** *allE*, *simp*)
  apply (*erule-tac* $x{=}x$ **in** *allE*, *erule-tac* $x{=}[a,\ b,\ c]$ **in** *allE*, *simp*)
  done

Thm 3.19 (3.62)

lemma *gyro-covariant-3-parallelogram*:
  shows $gyro\text{-}covariant\text{-}3\ (\lambda\ a\ b\ c.\ (b \oplus_c c) \ominus_b a)$
  unfolding *gyro-covariant-3-def*
proof *safe*
  fix $\tau\ a\ b\ c$
  assume $gyroaut\ \tau$
  then show $\tau\ ((b \oplus_c c) \ominus_b a) = (\tau\ b \oplus_c \tau\ c) \ominus_b \tau\ a$
   by (*smt* (*verit*, *ccfv-threshold*) *cogyroaut-def gyro-coaut-iff-gyro-aut local.gyro-left-cancel' local.gyro-left-inv local.gyroaut-def local.gyrominus-def*)
next
  fix $x\ a\ b\ c$
  have $((x \oplus b) \oplus_c (x \oplus c)) \ominus_b (x \oplus a) = (x \oplus b) \oplus gyr\ (x \oplus b)\ (\ominus\ (x \oplus c))\ ((x \oplus c) \ominus_b (x \oplus a))$
    by (*simp add*: *gyrominus-def mixed-gyroassoc-law*)
  also have ... $= (x \oplus b) \oplus gyr\ (x \oplus b)\ (\ominus\ (x \oplus c))\ (gyr\ x\ c\ (c \ominus_b a))$
    by (*simp add*: *gyro-translation-2b*)

also have ... $= x \oplus (b \oplus gyr\ b\ x\ (gyr\ (x \oplus b)\ (\ominus\ (x \oplus c))\ (gyr\ x\ c\ (c \ominus_b a))))$
using *local.gyro-right-assoc* **by** *auto*
also have ... $= x \oplus (b \oplus gyr\ b\ x\ (gyr\ x\ b\ (gyr\ b\ (\ominus\ c)\ (gyr\ c\ x\ (gyr\ x\ c\ (c \ominus_b a))))))$
unfolding *gyrominus-def*
using *gyro-commute-misc-4*[*of* $\ominus$ *x b c*]
by (*simp add*: *gyroautomorphic-inverse local.gyr-even*)
also have ... $= x \oplus (b \oplus gyr\ b\ (\ominus\ c)\ (c \ominus_b a))$
by (*metis local.gyr-auto-id2 local.gyr-right-loop pointfree-idE*)
finally show $x \oplus ((b \oplus_c c) \ominus_b a) = ((x \oplus b) \oplus_c (x \oplus c)) \ominus_b (x \oplus a)$
using *gyrominus-def mixed-gyroassoc-law*
by *auto*
qed

lemma *gyro-commute-misc6′*:
shows $x \oplus ((b \oplus_c c) \ominus_b a) = ((x \oplus b) \oplus_c (x \oplus c)) \ominus_b (x \oplus a)$
using *gyro-covariant-3-parallelogram*
unfolding *gyro-covariant-3-def*
by *simp*

(3.66)

lemma *gyro-commute-misc6*:
shows $(x \oplus b) \oplus_c (x \oplus c) = x \oplus ((b \oplus_c c) \oplus x)$
using *gyro-commute-misc6′*[*of x b c* $\ominus$ *x*]
by (*simp add*: *gyrominus-def*)

(3.67)

lemma *gyro-commute-misc6′′*:
shows $(x \oplus b) \oplus_c (x \ominus_b b) = x \oplus x$
using *gyro-commute-misc6 cogyro-gyro-inv cogyro-right-inv gyro-left-id gyrominus-def*
by *presburger*

end

type-synonym *′a rooted-gyrovec* = *′a* × *′a*

context *gyrogroup*
begin

Def 5.2.

fun *head* :: *′a rooted-gyrovec* ⇒ *′a* **where**
head (*p*, *q*) = *q*
fun *tail* :: *′a rooted-gyrovec* ⇒ *′a* **where**
tail (*p*, *q*) = *p*
fun *val* :: *′a rooted-gyrovec* ⇒ *′a* **where**
val $(p, q) = \ominus\ p \oplus q$
definition *ort* :: *′a* ⇒ *′a rooted-gyrovec* **where**
ort $p = (0_g, p)$

```
fun equiv-rooted-gyro-vec (infixl ∼ 100) where
  (p, q) ∼ (p′, q′) ⟷ ⊖p ⊕ q = ⊖p′ ⊕ q′

lemma equivp-equiv-rooted-gyro-vec [simp]:
  shows equivp (∼)
  unfolding equivp-def
  by fastforce

end
```

Def 5.4.

```
quotient-type (overloaded) ′a gyrovec = ′a :: gyrogroup × ′a / equiv-rooted-gyro-vec
  by auto

lift-definition vec :: ′a::gyrogroup ⇒ ′a ⇒ ′a gyrovec is λ p q. (p, q) .

definition ort :: ′a::gyrogroup ⇒ ′a gyrovec where
  ort A = vec 0_g A

context gyrocommutative-gyrogroup
begin
```

Thm 5.5. (5.4)

```
lemma equiv-rooted-gyro-vec-ex-t:
  shows (p, q) ∼ (p′, q′) ⟷ (∃ t. p′ = gyr p t (t ⊕ p) ∧ q′ = gyr p t (t ⊕ q))
(is ?lhs ⟷ ?rhs)
proof−
  let ?t = ⊖p ⊕ p′

  have ⊖(?t ⊕ p) ⊕ (?t ⊕ q) = gyr ?t p (⊖p ⊕ q)
    by (metis gyr-def gyro-equation-right)

  hence ⊖p ⊕ q = gyr (⊖ p) (⊖ ?t) (⊖(?t ⊕ p) ⊕ (?t ⊕ q))
    by (metis gyr-aut-inv-2 gyr-auto-id1 gyr-even inv-gyr-sym pointfree-idE)
  hence ⊖p ⊕ q = gyr p ?t (⊖(?t ⊕ p) ⊕ (?t ⊕ q))
    using gyr-even by presburger

  show ?thesis
  proof
    assume (p, q) ∼ (p′, q′)
    hence ∗: ⊖p ⊕ q = ⊖p′ ⊕ q′
      by simp

    have p′ = gyr p ?t (?t ⊕ p)
      using ∗
      using gyro-commute gyro-equation-right gyro-left-cancel by blast

    moreover
```

```
    have q′ = gyr p ?t (?t ⊕ q)
    proof−
      have q′ = p′ ⊕ (⊖ p ⊕ q)
        by (metis ∗ gyro-left-cancel′)
      also have ... = gyr p ?t (?t ⊕ p) ⊕ (⊖ p ⊕ q)
        using ‹p′ = gyr p (⊖ p ⊕ p′) (⊖ p ⊕ p′ ⊕ p)›
        by auto
      also have ... = (gyr p ?t ?t ⊕ gyr p ?t p) ⊕ (⊖ p ⊕ q)
        by simp
      also have ... = gyr p ?t ?t ⊕ (gyr p ?t p ⊕ gyr (gyr p ?t p) (gyr p ?t ?t) (⊖
p ⊕ q))
        using gyro-right-assoc by blast
      also have ... = gyr p ?t ?t ⊕ (gyr p ?t p ⊕ gyr p ?t (⊖ p ⊕ q))
        using gyr-commute-misc-1
        by presburger
      also have ... = gyr p ?t ?t ⊕ gyr p ?t q
        by (metis gyr-distrib gyro-equation-right)
      finally show q′ = gyr p ?t (?t ⊕ q)
        by simp
    qed

    ultimately

    show ?rhs
      by blast

  next

    assume ?rhs
    then obtain t where t: p′ = gyr p t (t ⊕ p) ∧ q′ = gyr p t (t ⊕ q)
      by auto

    have ⊖ p ⊕ q = gyr p t (⊖ (t ⊕ p) ⊕ (t ⊕ q))
      by (metis gyro-left-assoc gyro-left-cancel gyro-right-assoc gyro-translation-2a)
    also have ... = ⊖ (gyr p t (t ⊕ p)) ⊕ gyr p t (t ⊕ q)
      by (simp add: gyr-inv-3)
    finally show ?lhs
      using t
      by auto
  qed
qed
```

Thm 5.5. (5.5)

```
lemma gyro-translate-commute:
  assumes p′ = gyr p t (t ⊕ p) ∧ q′ = gyr p t (t ⊕ q)
  shows t = ⊖p ⊕ p′
  using assms
  using gyro-commute gyro-equation-right by blast
```

Def 5.6.

fun *gyrovec-translation* :: $'a \Rightarrow 'a$ *rooted-gyrovec* $\Rightarrow 'a$ *rooted-gyrovec* **where**
gyrovec-translation t $(p,\ q) = (gyr\ p\ t\ (t \oplus p),\ gyr\ p\ t\ (t \oplus q))$
end

lift-definition *gyrovec-translation'* :: $('a::gyrocommutative\text{-}gyrogroup)$ *gyrovec* $\Rightarrow$ $'a$ *rooted-gyrovec* $\Rightarrow 'a$ *rooted-gyrovec* **is**
$\lambda\ (tp,\ tq)\ (p,\ q).$ *gyrovec-translation* $(\ominus\ tp \oplus tq)\ (p,\ q)$
by *force*

(5.14)

lemma
shows *tail* (*gyrovec-translation* t $(p,\ q)) = p \oplus t$
by (*metis gyrovec-translation.simps gyro-commute tail.simps*)

(5.15)

lemma *gyrovec-translation-id*:
shows *gyrovec-translation* 0_g $(p,\ q) = (p,\ q)$
by *simp*

Thm 5.7.

lemma *equiv-rooted-gyrovec-t*:
shows $(p,\ q) \sim (p',\ q') \longleftrightarrow (p',\ q') =$ *gyrovec-translation* $(\ominus p \oplus p')\ (p,\ q)$
using *equiv-rooted-gyro-vec-ex-t gyro-translate-commute*
by (*metis gyrovec-translation.simps*)

Thm 5.8.

lemma *gyrovec-translation-head*:
assumes $(p',\ x) =$ *gyrovec-translation* t $(p,\ q)$
shows $x = p' \oplus (\ominus p \oplus q)$
by (*metis assms equiv-rooted-gyro-vec-ex-t gyrovec-translation.simps equiv-rooted-gyro-vec.simps gyro-equation-right*)

(5.24)

context *gyrocommutative-gyrogroup*
begin

definition *gyrovec-translation-inv'* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
gyrovec-translation-inv' $p\ t = \ominus\ (gyr\ p\ t\ t)$

lemma *gyrovec-translation-inv'*:
shows *gyrovec-translation* (*gyrovec-translation-inv'* $p\ t$) (*gyrovec-translation* t $(p,\ q)) = (p,\ q)$
unfolding *gyrovec-translation-inv'-def*
proof −
have *f1*: $\forall\ a\ aa.\ gyr\ (aa::'a)\ a\ (a \oplus aa) = aa \oplus a$
by (*metis* (*no-types*) *cogyro-commute cogyro-commute-iff-gyrocommute*)
have $\forall\ a\ aa.\ \ominus\ (gyr\ (aa::'a)\ a\ a) = \ominus\ (aa \oplus a) \oplus aa$
by (*simp add*: *gyr-misc-3*)

```
  then have ∀ a aa ab. (ab::'a, ab ⊕ (⊖ (ab ⊕ aa) ⊕ a)) = gyrovec-translation (⊖
(gyr ab aa aa)) (ab ⊕ aa, a)
   using f1 by (metis gyrovec-translation.simps gyr-commute-misc-3 gyro-left-cancel')
  then have ∀ a aa ab. gyrovec-translation (⊖ (gyr (ab::'a) aa aa)) (gyrovec-translation
aa (ab, a)) = (ab, a)
   using f1 by (metis gyrovec-translation.simps gyr-commute-misc-3 gyro-left-cancel')
  then show gyrovec-translation (⊖ (gyr p t t)) (gyrovec-translation t (p, q)) =
(p, q)
    by blast
qed

definition gyrovec-translation-compose' :: 'a ⇒ 'a ⇒ 'a ⇒ 'a where
  gyrovec-translation-compose' p t1 t2 = t1 ⊕ gyr t1 p t2

lemma gyrovec-translation-compose':
  gyrovec-translation t2 (gyrovec-translation t1 (p, q)) =
       gyrovec-translation (gyrovec-translation-compose' p t1 t2) (p, q)
  by (smt (verit) comp-eq-dest-lhs gyrovec-translation-compose'-def local.gyr-auto-id2
local.gyr-commute-misc-3 local.gyr-distrib local.gyr-nested-1 local.gyr-right-loop lo-
cal.gyro-commute local.gyro-right-assoc local.gyrovec-translation.simps pointfree-idE
prod.inject)

fun equiv-translate (infixl ~t 100) where
  (p1, q1) ~t (p2, q2) ⟷ (∃ t. gyrovec-translation t (p1, q1) = (p2, q2))

lemma equivp-equiv-translate:
  equivp (~t)
proof (rule equivpI)
  show reflp (~t)
  proof
    fix x
    show x ~t x
       by (metis equiv-translate.elims(3) gyr-commute-misc-3 gyr-id-2 gyro-left-id
gyrovec-translation.simps)
  qed
next
  show symp (~t)
  proof
    fix a b
    assume a ~t b
    thus b ~t a
      using gyrovec-translation-inv'
      by (cases a, cases b, fastforce)
  qed
next
  show transp (~t)
  proof
    fix x y z
    assume x ~t y y ~t z
```

```
    thus x ~t z
      using gyrovec-translation-compose'
      by (cases x, cases y, cases z, fastforce)
  qed
qed
```

(5.39)

```
definition vec :: 'a ⇒ 'a ⇒ 'a where
  vec a b = ⊖ a ⊕ b
```

(5.40)

```
lemma vec 0g b = b
  by (simp add: vec-def)
```

(5.41)

```
lemma
  assumes vec a b = v
  shows b = a ⊕ v
  by (metis assms local.gyro-left-cancel' local.vec-def)
```

(5.42)

```
lemma
  (a ⊕ v) ⊕ u = a ⊕ (v ⊕ gyr v a u)
  by (rule gyro-right-assoc)
```

(5.43)

```
lemma
  assumes vec a b = v
  shows a = ⊖v ⊕c b
  using assms
  using cogyro-commute cogyrominus-def gyro-equation-left gyro-left-cancel' vec-def
  by force

lemma
  shows (⊖ a ⊕ b) ⊕ gyr (⊖a) b (⊖ b ⊕ c) = ⊖a ⊕ c
  by (rule gyro-polygonal-addition-lemma)


definition torsion-elem::'a⇒ bool where
  torsion-elem g ⟷ g⊕g = 0g

end


class tf-tw-group = gyrocommutative-gyrogroup +
  assumes a1:∀ a. torsion-elem a ⟶ a = 0g
  assumes a2:∀ a. ∃ b. (b⊕b = a)
begin
```

T3.32

```
lemma unique-half:
  shows (a⊕a = c ∧ b⊕b = c) ⟶ a=b
proof
  assume (a⊕a = c ∧ b⊕b = c)
  show a=b
  proof−
    have a⊕a = b⊕b
      by (simp add: ‹a ⊕ a = c ∧ b ⊕ b = c›)
    moreover have ⊖ b ⊕ (a⊕a) = ⊖ b ⊕ (b⊕b)
      by (simp add: ‹a ⊕ a = c ∧ b ⊕ b = c›)
    moreover have ⊖ b ⊕ a  ⊕ (gyr (⊖ b) a) a =b
     by (metis ‹a ⊕ a = c ∧ b ⊕ b = c› local.gyro-left-assoc local.gyro-left-cancel')
    moreover have ⊖ b ⊕ a = ⊖ (⊖ b ⊕ a)
        by (metis calculation(3) local.gyro-plus-def-co local.gyro-right-cancel'-dual
local.gyroautomorphic-inverse)
    moreover have ⊖ b ⊕ a  ⊕ ( ⊖ b ⊕ a) =  0g
      by (metis calculation(4) local.gyro-rigth-inv)
    moreover have  torsion-elem (⊖ b ⊕ a)
      using calculation(5) local.torsion-elem-def by blast
    moreover have ( ⊖ b ⊕ a) = 0g
      using a1
      using calculation(6) by blast
    ultimately show ?thesis
      by (metis local.gyro-left-inv local.oplus-ominus-cancel)
  qed
qed
```

T3.33

```
lemma unique-gyro-half:
  assumes gh⊕gh = g
      gyr-h ⊕ gyr-h = gyr a b g
    shows gyr a b gh = gyr-h
  by (metis assms(1) assms(2) local.gyr-distrib unique-half)
```

3.102

```
lemma gh-minus:
  assumes gh⊕gh = ⊖ g
        gh2⊕gh2 = g
      shows  ⊖ gh2 = gh
   by (metis assms(1) assms(2) local.gyroautomorphic-inverse local.gyrominus-def
unique-half)
```

T3.34

```
lemma gyration-exclusion:
  assumes ∃ g. g≠ 0g
  shows ∀ a b.  gyr a b ≠  ⊖ ∘ id
proof(rule ccontr)
  assume ¬ (∀ a b.  gyr a b ≠  ⊖ ∘ id)
  have ∃ a b.( gyr a b =  ⊖ ∘ id)
```

```
    using ‹¬ (∀ a b. gyr a b ≠ ⊖ ∘ id)› by auto

  moreover obtain a b where gyr a b  = ⊖  ∘ id
    using calculation by blast
  moreover obtain bh where bh ⊕ bh = b
    using a2
    by blast
  moreover have a ⊕ (b ⊖b bh) = (a  ⊕ b)  ⊕ bh
  proof−
    have a ⊕ (b ⊖b bh) =  (a  ⊕ b) ⊖b  gyr a b bh
      by (simp add: local.gyr-inv-3 local.gyro-left-assoc local.gyrominus-def)
    moreover obtain gh where gh ⊕ gh =  gyr a b b
      using local.a2 by blast
    moreover have a ⊕ (b ⊖b bh) = (a  ⊕ b) ⊖b  gh
      using ‹bh ⊕ bh = b› calculation(1) calculation(2) unique-gyro-half by blast
    ultimately show ?thesis
      by (simp add: ‹gyr a b = ⊖ ∘ id› local.gyrominus-def)
  qed
  moreover have a ⊕ (b ⊖b bh) = a  ⊕ bh
    using calculation(3) local.gyro-left-right-cancel by force
  moreover have b =  0g
    by (metis calculation(4) calculation(5) local.cogyro-gyro-inv local.cogyroinv-def
local.gyro-equation-left local.gyro-equation-right)
  moreover have gyr a b = id
    by (simp add: calculation(6))
  moreover have gyr a b  = ⊖  ∘ id
    using calculation(2) by blast
  ultimately show False
    by (metis assms comp-id id-def local.gyro-rigth-inv unique-gyro-half)
qed
```

T3.35

```
lemma gyration-exclusion-cons:
  shows gyr a b b =  ⊖ b ⟶ b = 0g
proof
  assume gyr a b b =  ⊖ b
  show b = 0g
  proof−
   obtain bh where bh ⊕ bh = b
    using a2
    by blast
  moreover have a ⊕ (b ⊖b bh) = (a  ⊕ b)  ⊕ bh
  proof−
    have a ⊕ (b ⊖b bh) =  (a  ⊕ b) ⊖b  gyr a b bh
      by (simp add: local.gyr-inv-3 local.gyro-left-assoc local.gyrominus-def)
    moreover obtain gh where gh ⊕ gh =  gyr a b b
      using local.a2 by blast
    moreover have a ⊕ (b ⊖b bh) = (a  ⊕ b) ⊖b  gh
      using ‹bh ⊕ bh = b› calculation(1) calculation(2) unique-gyro-half by blast
```

ultimately show *?thesis*
by (*metis* ‹$bh \oplus bh = b$› ‹$gyr\ a\ b\ b = \ominus\ b$› *gh-minus local.gyro-inv-idem local.gyrominus-def*)
qed
moreover have $a \oplus (b \ominus_b bh) = a \oplus bh$
using *calculation*(*1*) *local.gyro-left-right-cancel* **by** *force*
ultimately show *?thesis*
by (*metis local.gyr-right-loop local.gyro-commute local.gyro-left-cancel local.gyro-right-id*)
qed
qed

T3.36

lemma *equation-t3-36*:
shows $x \ominus_b (y \ominus_b x) = y \longleftrightarrow x = y$
by (*metis gyration-exclusion-cons local.gyr-commute-misc-3 local.gyr-misc-1 local.gyr-misc-3 local.gyro-right-id local.gyro-rigth-inv local.gyrominus-def*)

end

locale *gyrogroup-isomorphism* =
fixes φ :: *'a::gyrocommutative-gyrogroup* ⇒ *'b*
fixes *gyrozero'* :: *'b* (0_{g1})
fixes *gyroplus'* :: *'b* ⇒ *'b* ⇒ *'b* (**infixl** $\oplus_1$ *100*)
fixes *gyroinv'* :: *'b* ⇒ *'b* ($\ominus_1$)
assumes *φzero* [*simp*]: $\varphi\ 0_g = 0_{g1}$
assumes *φplus* [*simp*]: $\varphi\ (a \oplus b) = (\varphi\ a) \oplus_1 (\varphi\ b)$
assumes *φminus* [*simp*]: $\varphi\ (\ominus\ a) = \ominus_1\ (\varphi\ a)$
assumes *φbij* [*simp*]: *bij* φ
begin

definition *gyr'* **where**
$gyr'\ a\ b\ x = \ominus_1\ (a \oplus_1 b) \oplus_1 (a \oplus_1 (b \oplus_1 x))$

lemma *φgyr* [*simp*]:
shows $\varphi\ (gyr\ a\ b\ z) = gyr'\ (\varphi\ a)\ (\varphi\ b)\ (\varphi\ z)$
by (*simp add*: *gyr'-def gyr-def*)

end

sublocale *gyrogroup-isomorphism* ⊆ *gyrogroupoid gyrozero' gyroplus'*
by *unfold-locales*

sublocale *gyrogroup-isomorphism* ⊆ *gyrocommutative-gyrogroup gyrozero' gyroplus' gyroinv' gyr'*
proof
fix *a*
show $0_{g1} \oplus_1 a = a$

```
    by (metis φbij φplus φzero bij-iff gyro-left-id)
next
  fix a
  show ⊖₁ a ⊕₁ a = 0g1
    by (metis φbij φminus φplus φzero bij-iff gyro-left-inv)
next
  fix a b z
  show a ⊕₁ (b ⊕₁ z) = a ⊕₁ b ⊕₁ gyr′ a b z
    using φgyr[of inv φ a inv φ b inv φ z]
    by (metis φbij φplus bij-inv-eq-iff gyro-left-assoc)
next
  fix a b
  show gyr′ a b = gyr′ (a ⊕₁ b) b
  proof
    fix z
    show gyr′ a b z = gyr′ (a ⊕₁ b) b z
      using φgyr[of inv φ a inv φ b inv φ z]
      by (smt (z3) φbij φgyr φplus bij-inv-eq-iff gyr-aut-inv-3 inv-gyr-sym)
  qed
next
  fix a b
  have ∗: gyr′ a b = φ ∘ (gyr (inv φ a) (inv φ b)) ∘ (inv φ)
  proof
    fix z
    show gyr′ a b z = (φ ∘ gyr (inv φ a) (inv φ b) ∘ inv φ) z
      by (metis φbij φgyr bij-inv-eq-iff comp-def)
  qed
  show gyroaut (gyr′ a b)
    unfolding gyroaut-def
  proof safe
    show bij (gyr′ a b)
      using ∗ φbij gyr-gyroaut
      by (metis bij-comp bij-imp-bij-inv gyrogroupoid-class.gyroaut-def)
  next
    fix x y
    show gyr′ a b (x ⊕₁ y) = gyr′ a b x ⊕₁ gyr′ a b y
      using ∗
      by (smt (verit, del-insts) φbij φplus bij-inv-eq-iff comp-def gyr-distrib)
  qed
next
  fix a b
  show a ⊕₁ b = gyr′ a b (b ⊕₁ a)
    using φgyr[of inv φ a inv φ b inv φ (b ⊕₁ a)]
    by (metis (no-types, lifting) φbij φplus bij-inv-eq-iff gyro-commute)
qed

end
theory More-Real-Vector
  imports Main HOL−Analysis.Inner-Product HOL.Real-Vector-Spaces
```

begin

lemma (**in** *real-vector*) *inner-eq-1*:
  assumes *norm a* = *1 norm b* = *1 inner a b* = *1*
  shows *a* = *b*
proof−
  have (*norm* (*a* − *b*))^2 = *inner* (*a* − *b*) (*a* − *b*)
    by (*simp add*: *norm-eq-sqrt-inner*)
  also have ... = *inner a a* − *2* ∗ (*inner a b*) + *inner b b*
    by (*simp add*: *inner-commute inner-diff-right*)
  also have ... = *0*
    using *assms*
    by (*simp add*: *norm-eq-sqrt-inner*)
  finally have *norm* (*a* − *b*) = *0*
    by *simp*
  thus *a* = *b*
    by *simp*
qed

end
theory *GyroVectorSpace*
 imports *GyroGroup HOL*−*Analysis.Inner-Product HOL.Real-Vector-Spaces More-Real-Vector*
begin

locale *gyrocarrier'* =
  fixes *to-carrier* :: *'a*::*gyrocommutative-gyrogroup* ⇒ *'b*::{*real-inner*}
  assumes *inj-to-carrier* [*simp*]: *inj to-carrier*
  assumes *to-carrier-zero* [*simp*]: *to-carrier* 0_g = *0*
begin

definition *carrier* :: *'b set* **where**
  carrier = *to-carrier* ‘ *UNIV*

lemma *bij-betw-to-carrier*:
  shows *bij-betw to-carrier UNIV carrier*
  by (*simp add*: *bij-betw-def carrier-def*)

definition *of-carrier* :: *'b* ⇒ *'a* **where**
  of-carrier = *inv to-carrier*

lemma *bij-betw-of-carrier*:
  shows *bij-betw of-carrier carrier UNIV*
  by (*simp add*: *carrier-def inj-imp-bij-betw-inv of-carrier-def*)

lemma *inj-on-of-carrier* [*simp*]:
  shows *inj-on of-carrier carrier*
  using *bij-betw-imp-inj-on bij-betw-of-carrier*
  by *auto*

lemma *to-carrier* [*simp*]:
shows $\bigwedge$ *b*. *b* $\in$ *carrier* $\Longrightarrow$ *to-carrier* (*of-carrier b*) = *b*
by (*simp add*: *carrier-def f-inv-into-f of-carrier-def*)

lemma *of-carrier* [*simp*]:
shows $\bigwedge$ *a*. *of-carrier* (*to-carrier a*) = *a*
by (*simp add*: *of-carrier-def*)

lemma *of-carrier-zero* [*simp*]:
shows *of-carrier* $0 = 0_g$
by (*simp add*: *inv-f-eq of-carrier-def*)

lemma *to-carrier-zero-iff*:
assumes *to-carrier a* = *0*
shows $a = 0_g$
using *assms*
by (*metis of-carrier to-carrier-zero*)

definition *gyronorm* :: *'a* ⇒ *real* (《-》 [*100*] *100*) **where**
《*a*》 = *norm* (*to-carrier a*)

definition *gyroinner* :: *'a* ⇒ *'a* ⇒ *real* (**infixl** · *100*) **where**
a · *b* = *inner* (*to-carrier a*) (*to-carrier b*)

lemma *norm-inner*: 《*a*》 = *sqrt* (*a* · *a*)
using *gyroinner-def gyronorm-def norm-eq-sqrt-inner* **by** *auto*

lemma *norm-zero*:
shows 《0_g》 = *0*
by (*simp add*: *gyronorm-def*)

lemma *norm-zero-iff*:
assumes 《*a*》 = *0*
shows $a = 0_g$
using *assms*
by (*simp add*: *gyronorm-def to-carrier-zero-iff*)

definition *norms* :: *real set* **where**
norms = {*x*. ∃ *a*. *x* = 《*a*》} ∪ {*x*. ∃ *a*. *x* = − 《*a*》}

lemma *norm-in-norms* [*simp*]:
shows 《*a*》 ∈ *norms*
by (*auto simp add*: *norms-def*)

lemma *minus-norm-in-norms* [*simp*]:
shows − 《*a*》 ∈ *norms*
by (*auto simp add*: *norms-def*)

end

locale *gyrocarrier* = *gyrocarrier′* +
  assumes *inner-gyroauto-invariant*: $\bigwedge$ *u v a b*. (*gyr u v a*) · (*gyr u v b*) = *a* · *b*
begin

lemma *norm-gyr*: ⟪*gyr u v a*⟫ = ⟪*a*⟫
  by (*metis inner-gyroauto-invariant norm-inner*)

end

locale *pre-gyrovector-space* = *gyrocarrier* +
  fixes *scale* :: *real* ⇒ ′*a* ⇒ ′*a* (**infixl** ⊗ *105*)
  assumes *scale-1*:
    $\bigwedge$ *a* :: ′*a*.
        1 ⊗ *a* = *a*
  assumes *scale-distrib*:
    $\bigwedge$ (*r1* :: *real*) (*r2* :: *real*) (*a* :: ′*a*).
        (*r1* + *r2*) ⊗ *a* = *r1* ⊗ *a* ⊕ *r2* ⊗ *a*
  assumes *scale-assoc*:
    $\bigwedge$ (*r1* :: *real*) (*r2* :: *real*) (*a* :: ′*a*).
        (*r1* ∗ *r2*) ⊗ *a* = *r1* ⊗ (*r2* ⊗ *a*)
  assumes *scale-prop1*:
    $\bigwedge$ (*r* :: *real*) (*a* :: ′*a*).
        ⟦*r* ≠ *0*; *a* ≠ 0_g⟧ ⟹ *to-carrier* (|*r*| ⊗ *a*) $/_R$ ⟪*r* ⊗ *a*⟫ = *to-carrier a* $/_R$ ⟪*a*⟫
  assumes *gyroauto-property*:
    $\bigwedge$ (*u* :: ′*a*) (*v* :: ′*a*) (*r* :: *real*) (*a* :: ′*a*).
        gyr u v (*r* ⊗ *a*) = *r* ⊗ (*gyr u v a*)
  assumes *gyroauto-id*:
    $\bigwedge$ (*r1* :: *real*) (*r2* :: *real*) (*v* :: ′*a*).
        gyr (*r1* ⊗ *v*) (*r2* ⊗ *v*) = *id*
begin

lemma *scale-minus1*:
  shows (−*1*) ⊗ *a* = ⊖ *a*
  by (*metis add.right-inverse add-cancel-right-left gyrogroup-class.gyro-left-cancel′ gyrogroup-class.gyro-right-id scale-1 scale-distrib*)

lemma *minus-norm*:
  shows ⟪⊖*a*⟫ = ⟪*a*⟫
 by (*smt* (*verit*, *best*) *gyro-inv-id of-carrier scale-minus1 inverse-eq-iff-eq of-carrier-zero scaleR-cancel-right scale-1 scale-prop1*)

  (6.3)

lemma *scale-minus*:
  shows (−*r*) ⊗ *a* = ⊖ (*r* ⊗ *a*)
  by (*metis minus-mult-commute mult-1 scale-assoc scale-minus1*)

lemma *scale-minus′*:
  shows *k* ⊗ (⊖ *a*) = ⊖ (*k* ⊗ *a*)

by (*metis mult.commute scale-assoc scale-minus1*)

lemma *zero-otimes* [*simp*]:
shows $0 \otimes x = 0_g$
by (*metis add.right-inverse gyro-rigth-inv scale-distrib scale-minus*)

lemma *times-zero* [*simp*]:
shows $t \otimes 0_g = 0_g$
by (*metis mult-zero-right scale-assoc zero-otimes*)

Theorem 6.4 (6.4)

lemma *monodistributive*:
shows $r \otimes (r1 \otimes a \oplus r2 \otimes a) = r \otimes (r1 \otimes a) \oplus r \otimes (r2 \otimes a)$
by (*metis ring-class.ring-distribs*(*1*) *scale-assoc scale-distrib*)

lemma *times2*: $2 \otimes a = a \oplus a$
by (*metis mult-2-right scale-1 scale-assoc scale-distrib*)

lemma *twosum*: $2 \otimes (a \oplus b) = a \oplus (2 \otimes b \oplus a)$
proof−
have $a \oplus (2 \otimes b \oplus a) = a \oplus ((b \oplus b) \oplus a)$
by (*simp add*: *times2*)
also have ... $= a \oplus (b \oplus (b \oplus gyr\ b\ b\ a))$
by (*simp add*: *gyro-right-assoc*)
also have ... $= a \oplus (b \oplus (b \oplus a))$
by *simp*
also have ... $= (a \oplus b) \oplus gyr\ a\ b\ (b \oplus a)$
using *gyro-left-assoc* **by** *blast*
also have ... $= (a \oplus b) \oplus (a \oplus b)$
by (*metis gyro-commute*)
finally show *?thesis*
by (*metis times2*)
qed

definition *gyrodistance* :: $'a \Rightarrow 'a \Rightarrow real$ ($d_\oplus$) **where**
$d_\oplus\ a\ b = \langle\langle \ominus\ a \oplus b \rangle\rangle$

lemma $d_\oplus\ a\ b = \langle\langle b \ominus_b a \rangle\rangle$
by (*metis gyrodistance-def gyrogroup-class.gyrominus-def gyro-commute norm-gyr*)

lemma *gyrodistance-metric-nonneg*:
shows $d_\oplus\ a\ b \geq 0$
by (*simp add*: *gyrodistance-def gyronorm-def*)

lemma *gyrodistance-metric-zero-iff*:
shows $d_\oplus\ a\ b = 0 \longleftrightarrow a = b$
unfolding *gyrodistance-def gyronorm-def*
by (*metis gyrogroup-class.gyro-left-cancel′ gyrogroup-class.gyro-right-id gyronorm-def norm-zero-iff real-normed-vector-class.norm-zero to-carrier-zero*)

lemma *gyrodistance-metric-sym*:
shows $d_\oplus\ a\ b = d_\oplus\ b\ a$
by (*metis gyrodistance-def gyrogroup-class.gyro-inv-idem gyrogroup-class.gyrominus-def gyrogroup-class.gyroplus-inv minus-norm norm-gyr*)

lemma *equation-solving*:
assumes $x \oplus y = a \ominus x \oplus y = b$
shows $x = (1/2) \otimes (a \ominus_{cb} b) \wedge y = (1/2) \otimes (a \ominus_{cb} b) \oplus b$
proof−
have $y = x \oplus b$
using *assms*(*2*) *gyro-equation-right* **by** *auto*
then have $a = x \oplus (x \oplus b)$
using *assms*(*1*) **by** *auto*
then have $a = (2 \otimes x) \oplus b$
by (*simp add*: *gyro-left-assoc times2*)
then have $x = (1/2) \otimes (a \ominus_{cb} b)$
by (*smt* (*verit*) *gyro-equation-left nonzero-eq-divide-eq scale-1 scale-assoc*)
then show *?thesis*
using ‹$y = x \oplus b$›
by *simp*
qed

lemma *double-plus*: $(2 \otimes a) \oplus b = a \oplus (a \oplus b)$
by (*simp add*: *gyro-left-assoc times2*)

lemma *I6-33*:
shows $(1/2) \otimes (a \ominus_{cb} b) = (-1/2) \otimes (b \ominus_{cb} a)$
by (*metis* (*no-types*, *opaque-lifting*) *div-by-1 divide-divide-eq-right divide-minus1 gyrogroup-class.gyro-equation-left gyrogroup-class.gyro-left-cancel′ scale-assoc scale-minus1 times-divide-eq-left*)

lemma *I6-34*:
shows $(1/2) \otimes (a \ominus_{cb} b) \oplus b = (1/2) \otimes (b \ominus_{cb} a) \oplus a$
by (*smt* (*verit*, *ccfv-threshold*) *I6-33 cogyro-right-cancel′ double-plus gyro-left-cancel′ mult.commute nonzero-eq-divide-eq scale-1 scale-assoc scale-minus1*)

lemma *I6-35*:
shows $gyr\ b\ a = gyr\ b\ ((1/2) \otimes (a \ominus_{cb} b) \oplus b) \circ (gyr\ ((1/2) \otimes (a \ominus_{cb} b) \oplus b)\ a)$
by (*metis* (*no-types*, *lifting*) *I6-33 I6-34 divide-minus-left gyr-master-misc2′ gyr-misc-2 gyr-right-loop gyro-commute gyro-translate-commute scale-minus*)

lemma *double-half*:
shows $2 \otimes ((1\ /\ 2) \otimes a) = a$
by (*metis field-sum-of-halves mult.commute mult-2-right scale-1 scale-assoc*)

lemma *I6-38*:
shows $a \oplus (1/2) \otimes (\ominus\ a \oplus_c b) = (1/2) \otimes (a \oplus b)$

```
proof −
  have ⋀r r′ a. r ⊗ (r′ ⊗ a) = r′ ⊗ (r ⊗ a)
    by (metis (full-types) mult.commute scale-assoc)
  then show ?thesis
    using double-half
   by (smt (z3) gyrogroup-class.cogyro-commute-iff-gyrocommute gyrogroup-class.cogyro-right-cancel′
gyrogroup-class.cogyrominus-def gyro-commute twosum)
qed

lemma I6-39:
  shows a ⊕ (1/2) ⊗ (⊖ a ⊕ b) = (1/2) ⊗ (a ⊕c b)
  by (metis I6-38 gyro-equation-right gyro-inv-idem)

lemma I6-40:
  shows gyr ((r + s) ⊗ a) b x = gyr (r⊗a) (s⊗a ⊕ b) (gyr (s⊗a) b x)
 by (metis (mono-tags, opaque-lifting) comp-eq-elim gyroauto-id id-def gyr-nested-1
scale-distrib)


definition collinear :: ′a => ′a => ′a => bool where
  collinear x y z ⟷ (y = z ∨ (∃ t::real. (x = y ⊕ t ⊗ (⊖ y ⊕ z))))

lemma collinear-aab:
  shows collinear a a b
  by (metis collinear-def gyro-right-id gyro-rigth-inv scale-distrib scale-minus)

lemma collinear-bab:
  shows collinear b a b
  by (metis collinear-def gyro-equation-right scale-1)

lemma T6-20:
  assumes collinear p1 a b collinear p2 a b a ≠ b p1 ≠ p2
  shows ∀ x. (collinear x p1 p2 ⟶ collinear x a b)
proof safe
  obtain t1 where t1: p1 = a ⊕ t1 ⊗ (⊖ a ⊕ b)
    using ‹collinear p1 a b› ‹a ≠ b› collinear-def
    by auto
  obtain t2 where t2: p2 = a ⊕ t2 ⊗ (⊖ a ⊕ b)
    using ‹collinear p2 a b› ‹a ≠ b› collinear-def
    by blast

  fix x
  assume collinear x p1 p2
  show collinear x a b
  proof−
    obtain t where t: x = p1 ⊕ t ⊗ (⊖ p1 ⊕ p2)
      using ‹collinear x p1 p2› ‹p1 ≠ p2› collinear-def
      by blast
    have x = (a ⊕ t1 ⊗ (⊖ a ⊕ b)) ⊕ t ⊗ (⊖ (a ⊕ t1 ⊗ (⊖ a ⊕ b)) ⊕ (a ⊕ t2 ⊗
```

```
(⊖ a ⊕ b)))
      using t1 t2 t
      by simp
    then have x = (a ⊕ t1 ⊗ (⊖ a ⊕ b)) ⊕ t ⊗ gyr a (t1 ⊗ (⊖ a ⊕ b)) ((−t1 +
t2) ⊗ (⊖ a ⊕ b))
      by (smt (verit, best) gyr-def scale-distrib)
    then have x = (a ⊕ t1 ⊗ (⊖ a ⊕ b)) ⊕ gyr a (t1 ⊗ (⊖ a ⊕ b)) ((t∗(−t1 +
t2)) ⊗ (⊖ a ⊕ b))
      using gyroauto-property scale-assoc by presburger
    then have x = a ⊕ (t1 ⊗ (⊖ a ⊕ b) ⊕ ((t∗(−t1 + t2)) ⊗ (⊖ a ⊕ b)))
      by (simp add: gyro-left-assoc)
    then  have x = a ⊕ (t1 + t∗(−t1 + t2)) ⊗ (⊖ a ⊕ b)
      by (simp add: scale-distrib)
    then show ?thesis
      using collinear-def by blast
  qed
qed

lemma T6-20-1:
  assumes collinear p1 a b collinear p2 a b p1 ≠ p2 a ≠ b
  shows ∀ x. (collinear x a b ⟶ collinear x p1 p2)
proof safe
  obtain t1 where t1: p1 = a ⊕ t1 ⊗ (⊖ a ⊕ b)
    using ‹collinear p1 a b› ‹a ≠ b› collinear-def
    by auto
  obtain t3 where t3: p2 = a ⊕ t3 ⊗ (⊖ a ⊕ b)
    using ‹collinear p2 a b› ‹a ≠ b› collinear-def
    by blast

  fix x
  assume collinear x a b
  show collinear x p1 p2
  proof−
    obtain t2 where x = a ⊕ t2 ⊗ (⊖ a ⊕ b)
      using ‹collinear x a b› ‹a ≠ b› collinear-def
      by blast
    show ?thesis
    proof (cases t1 = t3)
      case True
      then show ?thesis
        using t1 t3 ‹p1 ≠ p2›
        by blast
    next
      case False
      then obtain t where t: t = (t2−t1)/(t3−t1)
        by simp
      have p1 ⊕ t ⊗ (⊖ p1 ⊕ p2) = (a ⊕ t1 ⊗ (⊖ a ⊕ b)) ⊕ t ⊗ (⊖ (a ⊕ t1 ⊗
(⊖ a ⊕ b)) ⊕ ( a ⊕ t3 ⊗ (⊖ a ⊕ b)))
        using t1 t3 by blast
```

then have $p1 \oplus t \otimes (\ominus p1 \oplus p2) = (a \oplus t1 \otimes (\ominus a \oplus b)) \oplus t \otimes gyr\ a\ (t1 \otimes (\ominus a \oplus b))\ (t1 \otimes (\ominus\ (\ominus a \oplus b)) \oplus t3 \otimes (\ominus a \oplus b))$
by (*metis* (*no-types*, *lifting*) *gyro-translation-2a mult.commute scale-assoc scale-minus1*)
then have $p1 \oplus t \otimes (\ominus p1 \oplus p2) = (a \oplus t1 \otimes (\ominus a \oplus b)) \oplus gyr\ a\ (t1 \otimes (\ominus a \oplus b))\ (((-t1+t3)*t) \otimes (\ominus a \oplus b)\ )$
by (*metis* (*no-types*, *opaque-lifting*) *gyroauto-property minus-mult-commute mult.commute mult.right-neutral scale-assoc scale-distrib scale-minus1*)
then have $p1 \oplus t \otimes (\ominus p1 \oplus p2) = a \oplus (t1 \otimes (\ominus a \oplus b) \oplus ((-t1+t3)*t) \otimes (\ominus a \oplus b))$
using *gyro-left-assoc* **by** *metis*
then have $p1 \oplus t \otimes (\ominus p1 \oplus p2) = a \oplus (t1 + (-t1+t3)*t) \otimes ((\ominus a \oplus b))$
using *scale-distrib* **by** *presburger*
moreover have $t1 + (-t1+t3)*t = t2$
using ‹$t1 \neq t3$› *t*
by *simp*
ultimately
have $p1 \oplus t \otimes (\ominus p1 \oplus p2) = a \oplus t2 \otimes (\ominus a \oplus b)$
by *blast*
then show *?thesis*
using ‹$x = a \oplus t2 \otimes (\ominus a \oplus b)$›
unfolding *collinear-def*
by *metis*
qed
qed
qed

lemma *collinear-sym1*:
assumes *collinear a b c*
shows *collinear b a c*
using *T6-20-1 assms collinear-aab collinear-bab collinear-def* **by** *blast*

lemma *collinear-sym2*:
assumes *collinear a b c*
shows *collinear a c b*
by (*metis T6-20 assms collinear-aab collinear-bab*)

lemma *collinear-transitive*:
assumes *collinear a b c collinear d b c* $b \neq c$
shows *collinear a d b*
by (*metis T6-20 assms*(*1*) *assms*(*2*) *assms*(*3*) *collinear-bab collinear-sym1 collinear-sym2*)

lemma *collinear-translate'*:
shows $x = u \oplus t \otimes (\ominus u \oplus v) \longleftrightarrow$
$(\ominus a \oplus x) = (\ominus a \oplus u) \oplus t \otimes (\ominus (\ominus a \oplus u) \oplus (\ominus a \oplus v))$
by (*metis* (*no-types*, *lifting*) *gyr-misc-2 gyro-right-assoc gyro-translation-2a gyroauto-property oplus-ominus-cancel*)

definition *translate* **where**

translate a x = ⊖ *a* ⊕ *x*

lemma *collinear-translate*:
shows *collinear u v w* ⟷ *collinear* (*translate a u*) (*translate a v*) (*translate a w*)
unfolding *collinear-def translate-def*
by (*metis collinear-translate′ gyro-left-cancel′*)

definition *gyroline* :: ′*a* ⇒ ′*a* ⇒ ′*a set* **where**
gyroline a b = {*x*. *collinear x a b*}

definition *between* :: ′*a* => ′*a* => ′*a* => *bool* **where**
between x y z ⟷ (∃ *t*::*real*. *0* ≤ *t* ∧ *t* ≤ *1* ∧ *y* = *x* ⊕ *t* ⊗ (⊖ *x* ⊕ *z*))

lemma *between-xxy* [*simp*]:
shows *between x x y*
unfolding *between-def* **by** *force*

lemma *between-xyy* [*simp*]:
shows *between x y y*
unfolding *between-def*
by (*rule exI* [**where** *x*=*1*]) (*simp add*: *scale-1*)

lemma *between-xyx*:
assumes *between x y x*
shows *y* = *x*
using *assms*
unfolding *between-def*
by *auto*

lemma *between-translate*:
shows *between u v w* ⟷ *between* (*translate a u*) (*translate a v*) (*translate a w*)
unfolding *between-def translate-def*
using *collinear-translate′*
by *auto*

definition *distance* **where**
distance u v = ⟪⊖ *u* ⊕ *v*⟫

lemma *distance-translate*:
shows *distance u v* = *distance* (*translate a u*) (*translate a v*)
unfolding *distance-def translate-def*
using *gyro-translation-2a norm-gyr*
by *metis*

end

locale *gyrocarrier-norms-embed′* = *gyrocarrier′ to-carrier*
for *to-carrier* :: ′*a*::*gyrocommutative-gyrogroup* ⇒ ′*b*::{*real-inner*, *real-normed-algebra-1*}

+
  assumes *norms-carrier*: *of-real* ‘ *norms* ⊆ *carrier*
begin

definition *of-real′* :: *real* ⇒ *′a* **where**
  of-real′ = *of-carrier* ∘ *of-real*

definition *reals* :: *′a set* **where**
  reals = *of-carrier* ‘ *of-real* ‘ *norms*

lemma *bij-reals-norms*:
  shows *bij-betw of-real′ norms reals*
  unfolding *of-real′-def*
  by (*metis bij-betw-def comp-inj-on-iff dual-order.eq-iff image-comp inj-image-eq-iff inj-of-real inj-on-of-carrier norms-carrier reals-def subset-image-inj subset-inj-on*)

lemma *inj-on-of-real′*:
  shows *inj-on of-real′ norms*
  using *bij-betw-imp-inj-on bij-reals-norms*
  by *blast*

definition *to-real* :: *′b* ⇒ *real* **where**
  to-real = *the-inv-into norms of-real*

lemma *to-real* [*simp*]:
  assumes x ∈ *norms*
  shows *to-real* (*of-real* x) = x
  using *assms*
  by (*metis inj-on-imageI2 inj-on-of-real′ of-real′-def the-inv-into-f-f to-real-def*)

lemma *of-real* [*simp*]:
  assumes x ∈ *of-real* ‘ *norms*
  shows *of-real* (*to-real* x) = x
  using *assms*
  using *to-real*
  by *force*

definition *to-real′* :: *′a* ⇒ *real* **where**
  to-real′ = *to-real* ∘ *to-carrier*

lemma *bij-betw-to-real′*:
  bij-betw to-real′ reals norms
  by (*smt* (*verit*, *best*) *bij-betw-cong bij-betw-the-inv-into bij-reals-norms comp-apply f-the-inv-into-f to-carrier image-eqI in-mono inj-on-imageI inj-on-imageI2 inj-on-of-real′ norms-carrier of-real′-def reals-def the-inv-into-comp the-inv-into-onto to-real′-def to-real-def*)

lemma *to-real′* [*simp*]:
  assumes x ∈ *norms*

shows *to-real′* (*of-real′* x) = x
using *assms*
using *norms-carrier of-real′-def to-real′-def*
by *auto*

lemma *of-real′* [*simp*]:
assumes $x \in$ *reals*
shows *of-real′* (*to-real′* x) = x
by (*smt* (*verit*, *ccfv-SIG*) *assms bij-betw-iff-bijections bij-reals-norms to-real′*)

lemma *to-real′-norm* [*simp*]:
shows *to-real′* (*of-real′* (⟪a⟫)) = (⟪a⟫)
using *norms-def to-real′*
by *auto*

lemma *gyronorm-of-real′*:
assumes $x \in$ *norms*
shows ⟪*of-real′* x⟫ = *abs* x
unfolding *of-real′-def gyronorm-def comp-def*
proof (*subst to-carrier*)
show *of-real* $x \in$ *carrier*
using *assms norms-carrier* **by** *auto*
next
show *norm* ((*of-real*::*real*⇒′*b*) x) = $|x|$
using *norm-of-real*
by *auto*
qed

lemma *gyronorm-abs-to-real′*:
assumes $x \in$ *reals*
shows *abs* (*to-real′* x) = ⟪x⟫
using *assms*
by (*metis bij-betwE bij-betw-to-real′ gyronorm-of-real′ of-real′*)

definition *oplusR* :: *real* ⇒ *real* ⇒ *real* (**infixl** $\oplus_R$ *100*) **where**
$a \oplus_R b$ = *to-real′* (*of-real′* $a \oplus$ *of-real′* b)

definition *oinvR* :: *real* ⇒ *real* ($\ominus_R$) **where**
$\ominus_R\ a$ = *to-real′* ($\ominus$ (*of-real′* a))

end

locale *gyrocarrier-norms-embed* = *gyrocarrier-norms-embed′* +
fixes *scale* :: *real* ⇒ ′*a* ⇒ ′*a* (**infixl** ⊗ *105*)
assumes *oplus-reals*: $\bigwedge$ a b. ⟦$a \in$ *reals*; $b \in$ *reals*⟧ $\Longrightarrow$ $a \oplus b \in$ *reals*
assumes *oinv-reals*: $\bigwedge$ a. $a \in$ *reals* $\Longrightarrow$ $\ominus\ a \in$ *reals*
assumes *otimes-reals*: $\bigwedge$ a r. $a \in$ *reals* $\Longrightarrow$ $r \otimes a \in$ *reals*
begin

definition *otimesR* :: *real* ⇒ *real* ⇒ *real* (**infixl** $\otimes_R$ *100*) **where**
  $r \otimes_R a = \textit{to-real}'\ (r \otimes (\textit{of-real}'\ a))$

lemma *oplusR-norms*:
  shows $\bigwedge a\ b.\ [\![a \in \textit{norms};\ b \in \textit{norms}]\!] \Longrightarrow a \oplus_R b \in \textit{norms}$
  by (*metis bij-betwE bij-betw-to-real′ bij-reals-norms oplusR-def oplus-reals*)

lemma *oinvR-norms*:
  shows $\bigwedge a.\ a \in \textit{norms} \Longrightarrow \ominus_R\ a \in \textit{norms}$
  by (*metis bij-betwE bij-betw-to-real′ bij-reals-norms oinvR-def oinv-reals*)

lemma *otimesR-norms*:
  shows $\bigwedge a\ r.\ a \in \textit{norms} \Longrightarrow r \otimes_R a \in \textit{norms}$
  by (*metis bij-betwE bij-betw-to-real′ bij-reals-norms otimesR-def otimes-reals*)

lemma *of-real′-oplusR* [*simp*]:
  shows $\textit{of-real}'\ ((\langle\!\langle a\rangle\!\rangle) \oplus_R (\langle\!\langle b\rangle\!\rangle)) = (\textit{of-real}'\ (\langle\!\langle a\rangle\!\rangle) \oplus \textit{of-real}'\ (\langle\!\langle b\rangle\!\rangle))$
  unfolding *oplusR-def*
 by (*smt* (*verit*, *ccfv-threshold*) *Un-iff bij-betw-iff-bijections bij-reals-norms mem-Collect-eq norms-def of-real′ oplus-reals*)

lemma *of-real′-otimesR* [*simp*]:
  shows $\textit{of-real}'\ (r \otimes_R (\langle\!\langle a\rangle\!\rangle)) = r \otimes (\textit{of-real}'\ (\langle\!\langle a\rangle\!\rangle))$
  unfolding *otimesR-def*
 by (*metis* (*mono-tags*, *lifting*) *Un-iff bij-betwE bij-reals-norms norms-def mem-Collect-eq of-real′ otimes-reals*)

lemma *of-real′-oinvR* [*simp*]:
  shows $\textit{of-real}'\ (\ominus_R\ (\langle\!\langle a\rangle\!\rangle)) = \ominus\ (\textit{of-real}'\ (\langle\!\langle a\rangle\!\rangle))$
  unfolding *oinvR-def*
  by (*metis* (*mono-tags*, *lifting*) *Un-iff bij-betwE bij-reals-norms mem-Collect-eq norms-def of-real′ oinv-reals*)

end

locale *gyrovector-space-norms-embed* =
  gyrocarrier to-carrier +
  gyrocarrier-norms-embed to-carrier +
  pre-gyrovector-space to-carrier
 for *to-carrier* :: ′*a*::*gyrocommutative-gyrogroup* ⇒ ′*b*::{*real-inner*, *real-normed-algebra-1*}
+
  assumes *homogeneity*:
    $\bigwedge (r :: \textit{real})\ (a :: 'a).$
      $\langle\!\langle r \otimes a\rangle\!\rangle = |r| \otimes_R (\langle\!\langle a\rangle\!\rangle)$
  assumes *gyrotriangle*:
    $\bigwedge (a :: 'a)\ (b :: 'a).$
      $\langle\!\langle a \oplus b\rangle\!\rangle \le (\langle\!\langle a\rangle\!\rangle) \oplus_R (\langle\!\langle b\rangle\!\rangle)$
begin

lemma *gyrodistance-gyrotriangle*:
  shows $d_{\oplus}\ a\ c \leq d_{\oplus}\ a\ b\ \ \oplus_R\ d_{\oplus}\ b\ c$
proof−
  have $\langle\!\langle \ominus a \oplus c \rangle\!\rangle = \langle\!\langle (\ominus a \oplus b) \oplus gyr\ (\ominus a)\ b\ (\ominus b \oplus c) \rangle\!\rangle$
    using *gyro-polygonal-addition-lemma*[*of a b c*]
    by *auto*
  also have ... $\leq\ (\langle\!\langle \ominus a \oplus b \rangle\!\rangle) \oplus_R (\langle\!\langle gyr\ (\ominus a)\ b\ (\ominus b \oplus c) \rangle\!\rangle)$
    by (*simp add*: *gyrotriangle*)
  finally show *?thesis*
    unfolding *gyrodistance-def norm-gyr*
    by *meson*
qed

end

end
theory *VectorSpace*
  imports *Main HOL.Real HOL−Types-To-Sets.Linear-Algebra-On-With*

begin

locale *vector-space-with-domain* =
  fixes *dom* :: $'a$ *set*
    and *add* :: $'a \Rightarrow 'a \Rightarrow 'a$
    and *zero* :: $'a$
    and *smult* :: *real* $\Rightarrow 'a \Rightarrow 'a$
  assumes *add-closed*: $[\![x \in dom;\ y \in dom]\!] \Longrightarrow add\ x\ y \in dom$
    and *zero-in-dom*: $zero \in dom$
    and *add-assoc*: $[\![x \in dom;\ y \in dom;\ z\in dom]\!] \Longrightarrow add\ (add\ x\ y)\ z = add\ x\ (add$
$y\ z)$
    and *add-comm*: $[\![x \in dom;\ y \in dom]\!] \Longrightarrow add\ x\ y = add\ y\ x$
    and *add-zero*: $[\![x \in dom]\!] \Longrightarrow add\ x\ zero = x$
    and *add-inv*: $x \in dom \Longrightarrow \exists\, y \in dom.\ add\ x\ y = zero$
    and *smult-closed*: $[\![x \in dom]\!] \Longrightarrow smult\ a\ x \in dom$
    and *smult-distr-sadd*: $[\![x \in dom]\!] \Longrightarrow smult\ (a + b)\ x = add\ (smult\ a\ x)\ (smult$
$b\ x)$
    and *smult-assoc*: $[\![x \in dom]\!] \Longrightarrow smult\ a\ \ (smult\ b\ x) = smult\ (a * b)\ \ x$
    and *smult-one*: $[\![x \in dom]\!] \Longrightarrow smult\ 1\ x = x$
    and *smult-distr-sadd2*: $[\![x \in dom;\ y\in dom]\!] \Longrightarrow smult\ a\ (add\ x\ y) = add\ (smult$
$a\ x)\ (smult\ a\ y)$

begin

lemma *inv-unique*:
  assumes $a\in dom\ z1\in dom\ z2\in dom$
      $add\ a\ z1\ = zero$
      $add\ a\ z2\ = zero$
    shows $z1{=}z2$

```
  by (metis add-assoc add-comm add-zero assms)

definition inv::'a⇒'a where
    inv a = (if a∈dom then (THE z. (z∈dom ∧ add a z = zero)) else undefined)

definition minus::'a⇒'a⇒'a where
    minus a b = (if a∈dom ∧ b∈dom then add a (inv b) else undefined)

lemma module-on-with-is-this:
  shows module-on-with dom add minus inv zero smult
  unfolding module-on-with-def
proof
  show ab-group-add-on-with dom add zero local.minus local.inv
    unfolding ab-group-add-on-with-def
  proof
    show comm-monoid-add-on-with dom add zero
     by (smt (verit, del-insts) ab-semigroup-add-on-with-Ball-def add-assoc add-closed
add-comm add-zero comm-monoid-add-on-with-Ball-def semigroup-add-on-with-def
zero-in-dom)
    next
    show ab-group-add-on-with-axioms dom add zero local.minus local.inv
      unfolding ab-group-add-on-with-axioms-def
    proof
      show ∀ a. a ∈ dom ⟶ add (local.inv a) a = zero
      proof−
        {fix a
          assume a∈dom
          obtain z where z∈dom ∧ add z a = zero
            using ‹a ∈ dom› add-comm add-inv by blast
          moreover have local.inv a = z
            using ‹a ∈ dom› add-comm calculation inv-unique local.inv-def by auto
          ultimately have  add (local.inv a) a = zero
          using ‹a∈dom›
          by fastforce

      }
      show ?thesis
        using ‹⋀aa. aa ∈ dom ⟹ add (local.inv aa) aa = zero› by blast
    qed
  next
    show  (∀ a b. a ∈ dom ⟶ b ∈ dom ⟶ local.minus a b = add a (local.inv b))
∧
    (∀ a. a ∈ dom ⟶ local.inv a ∈ dom)
    proof
      show ∀ a b. a ∈ dom ⟶ b ∈ dom ⟶ local.minus a b = add a (local.inv b)
        using minus-def by auto
    next
      show ∀ a. a ∈ dom ⟶ local.inv a ∈ dom
      proof−
```

```
      {fix a
      assume a∈dom
      have local.inv a ∈dom
       by (smt (z3) ‹a ∈ dom› add-assoc add-comm add-inv add-zero local.inv-def
theI')
    }
    show ?thesis
      using ‹⋀aa. aa ∈ dom ⟹ local.inv aa ∈ dom› by fastforce
      qed
    qed
  qed
  qed
next
  show ((∀ a. ∀ x∈dom. ∀ y∈dom. smult a (add x y) = add (smult a x) (smult a
y)) ∧
     (∀ a b. ∀ x∈dom. smult (a + b) x = add (smult a x) (smult b x))) ∧
    (∀ a b. ∀ x∈dom. smult a (smult b x) = smult (a ∗ b) x) ∧
    (∀ x∈dom. smult 1 x = x) ∧ (∀ a. ∀ x∈dom. smult a x ∈ dom)
    using smult-one smult-distr-sadd smult-assoc smult-closed
      smult-distr-sadd2
    by auto
qed

lemma vector-space-on-with-is-this:
  shows vector-space-on-with dom add minus inv zero smult
  by (simp add: module-on-with-is-this vector-space-on-with-def)

end

end
theory Abe
  imports GyroGroup HOL−Analysis.Inner-Product HOL.Real-Vector-Spaces Vec-
torSpace
begin



locale one-dim-vector-space-with-domain =
  vector-space-with-domain +
  assumes ∀ y. ∀ x. (y∈ dom ∧
 x∈ dom ∧ x≠zero ⟶ (∃!r::real. y = smult r x))

locale GGV =
  fixes fi ::'a::gyrocommutative-gyrogroup ⇒ 'b::real-inner
  fixes scale ::real ⇒ 'a ⇒ 'a
  fixes plus'::real ⇒ real ⇒ real
  fixes smult'::real ⇒ real ⇒ real

  assumes inj fi
```

assumes *norm* (*fi* (*gyr u v a*)) = *norm* (*fi a*)
assumes *scale 1 a* = *a*
assumes *scale* (*r1*+*r2*) *a* = (*scale r1 a*) ⊕ (*scale r2 a*)
assumes *scale* (*r1*∗*r2*) *a* = *scale r1* (*scale r2 a*)
assumes (*a*≠*gyrozero* ∧ *r*≠*0*)⟶ (*fi* (*scale* |*r*| *a*)) $/_R$ (*norm* (*fi* (*scale r a*))) = (*fi a*) $/_R$ (*norm* (*fi a*))
assumes *gyr u v* (*scale r a*) = *scale r* (*gyr u v a*)
assumes *gyr* (*scale r1 v*) (*scale r2 v*) = *id*
assumes *vector-space-with-domain* {*x*.∃ *a*. *x* = *norm* (*fi a*) ∨ *x* = − *norm* (*fi a*)} *plus′ 0 smult′*
assumes *norm* (*fi* (*scale r a*)) = *smult′* |*r*| (*norm* (*fi a*))
assumes *norm* (*fi* (*a* ⊕ *b*)) = *plus′* (*norm* (*fi a*)) (*norm* (*fi b*))
begin

end

class *gyrolinear-space* =
gyrocommutative-gyrogroup +
fixes *scale* :: *real* ⇒ ′*a*::*gyrocommutative-gyrogroup* ⇒ ′*a* (**infixl** ⊗ *105*)
assumes *scale-1*: ⋀ *a* :: ′*a*. *1* ⊗ *a* = *a*
assumes *scale-distrib*: ⋀ (*r1* :: *real*) (*r2* :: *real*) (*a* :: ′*a*). (*r1* + *r2*) ⊗ *a* = *r1* ⊗ *a* ⊕ *r2* ⊗ *a*
assumes *scale-assoc*: ⋀ (*r1* :: *real*) (*r2* :: *real*) (*a* :: ′*a*). (*r1* ∗ *r2*) ⊗ *a* = *r1* ⊗ (*r2* ⊗ *a*)
assumes *gyroauto-property*: ⋀ (*u* :: ′*a*) (*v* :: ′*a*) (*r* :: *real*) (*a* :: ′*a*). *gyr u v* (*r* ⊗ *a*) = *r* ⊗ (*gyr u v a*)
assumes *gyroauto-id*: ⋀ (*r1* :: *real*) (*r2* :: *real*) (*v* :: ′*a*). *gyr* (*r1* ⊗ *v*) (*r2* ⊗ *v*) = *id*

begin

end

locale *normed-gyrolinear-space* =
fixes *norm′*::′*a*::*gyrolinear-space* ⇒ *real*
fixes *f*::*real* ⇒ *real*
assumes ∀ *a*::′*a*. (*norm′ a* ≥ *0*)
assumes ∀ *y*::*real*. (*y*∈ (*norm′* ‘ *UNIV*) ⟶ (*f y*) ≥ *0*)
assumes *bij-betw f* (*norm′* ‘ *UNIV*) {*x*::*real*. *x*≥*0*}
assumes ∀ *y*::*real*. ∀ *z*::*real*. ((*y*∈ *norm′* ‘ *UNIV* ∧
z∈ *norm′* ‘ *UNIV* ∧ *y*>*z*)⟶ (*f y*) > (*f z*))

assumes ∀ *x*::′*a*. ∀ *y*::′*a*. *f*(*norm′* (*gyroplus x y*)) ≤ (*f* (*norm′ x*)) + (*f* (*norm′ y*))
assumes *f* (*norm′* (*scale r x*)) = |*r*| ∗ (*f* (*norm′ x*))
assumes *norm′* (*gyr u v x*) = *norm′ x*
assumes ∀ *x*::′*a*. ((*norm′ x*) = *0* ⟷ *x* = *gyrozero*)
begin

definition *norms*::*real set* **where**
norms = *norm'* ‘ *UNIV*

definition *norms-neg*::*real set* **where**
norms-neg = (λx. -1 ∗ *norm' x*) ‘ *UNIV*

definition *norms-all*::*real set* **where**
norms-all = *norms* ∪ *norms-neg*

lemma *norms-neq-not-empty*:
shows *norms-neg* ≠ {}
using *add.inverse-inverse norms-neg-def* **by** *fastforce*

lemma *zero-only-norms-norms-neg*:
assumes *x*∈*norms x*∈*norms-neg*
shows *x*=*0*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*1*) *assms*(*2*) *f-inv-into-f normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def norms-neg-def*)

lemma *a1-a2*:
shows ∃ *f'*:: *real* ⇒ *real*. ((∀ *x*::*real*. ∀ *y*::*real*. (*x*∈*norms-all* ∧ *y* ∈*norms-all* ∧ *x*>*y*)⟶ (*f' x*) > (*f' y*))
∧ (*f' 0*) = *0* ∧ *bij-betw f' norms-all UNIV*)
proof−
let *?f'* = λx. *if x*=*0 then 0 else if* (*x* ∈ *norms*) *then* (*f x*) *else if* (*x*∈ *norms-neg*) *then* − (*f* (−*x*)) *else undefined*
have *fact3*: *?f' 0* = *0*
by *auto*
moreover have *fact1*: (∀ *x*::*real*. ∀ *y*::*real*. (*x*∈*norms-all* ∧ *y* ∈*norms-all* ∧ *x*>*y*)⟶ (*?f' x*) > (*?f' y*))
proof−
{**fix** *x y*
assume *x*∈*norms-all* ∧ *y* ∈*norms-all* ∧ *x*>*y*
have (*?f' x*) > (*?f' y*)
proof−
have *x*=*0* ∨ *x*≠*0* **by** *blast*
moreover {
assume *x*=*0*
then have *?thesis*
by (*smt* (*verit*, *del-insts*) *Un-def* ‹*x* ∈ *norms-all* ∧ *y* ∈ *norms-all* ∧ *y* < *x*› *f-inv-into-f mem-Collect-eq normed-gyrolinear-space.norms-neg-def normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-all-def norms-def rangeI*)

}
moreover {
assume *x*≠*0*
have *x*∈*norms* ∨ *x*∈*norms-neg*

```
      using ‹x ∈ norms-all ∧ y ∈ norms-all ∧ y < x› norms-all-def by force
    moreover {
      assume x∈norms
      then have y=0∨ y≠0
        by blast
      moreover {
        assume y=0
        then have ?thesis
          by (smt (z3) ‹x ∈ norms› ‹x ∈ norms-all ∧ y ∈ norms-all ∧ y < x›
normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def rangeI)
      } moreover {
        assume y≠0
        have y∈norms ∨ y∈norms-neg
          using ‹x ∈ norms-all ∧ y ∈ norms-all ∧ y < x› norms-all-def by auto
        moreover {
          assume y∈norms
          then have ?thesis
            by (smt (z3) ‹x ∈ norms› ‹x ∈ norms-all ∧ y ∈ norms-all ∧ y < x›
‹x ≠ 0› normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def)

        } moreover {
          assume y∈norms-neg
          then have ?f' y = − (f (−y))
            using ‹y ≠ 0› zero-only-norms-norms-neg by fastforce
          moreover have −y ∈ norms
            using ‹y ∈ norms-neg› norms-def norms-neg-def by force
          moreover have ?f' y ≤ 0

                  by (smt (verit, ccfv-threshold) calculation(1) calculation(2)
normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def)

          moreover have ?f' y ≠0
          proof(rule ccontr)
            assume ¬(?f' y ≠ 0)
            then show False
                by (smt (verit, del-insts) ‹y ≠ 0› calculation(1) calculation(2)
f-inv-into-f normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def
rangeI)
          qed
          ultimately have ?thesis
        by (smt (z3) ‹x ∈ norms› normed-gyrolinear-space-axioms normed-gyrolinear-space-def
norms-def)
        }
        ultimately have ?thesis by blast
        }
        ultimately have ?thesis by blast
      } moreover {
        assume x∈norms-neg
        then have ?thesis
```

```
          by (smt (verit, del-insts) Un-def ‹x ∈ norms-all ∧ y ∈ norms-all ∧ y < x›
f-inv-into-f mem-Collect-eq normed-gyrolinear-space.norms-neg-def normed-gyrolinear-space-axioms
normed-gyrolinear-space-def norms-all-def norms-def rangeI)

        }
        ultimately have ?thesis by blast
      } ultimately show ?thesis by blast
    qed
  }
  then show ?thesis by blast
qed
  moreover have fact2: bij-betw ?f' norms-all UNIV
  proof−
    have ∗:∀x. ∀y. (x∈norms-all ∧ y∈norms-all ∧ (?f' x) = (?f' y)) ⟶ x = y
      by (smt (verit, ccfv-threshold) calculation(2))
    moreover have ∗∗:∀x::real. ∃y. (y∈ norms-all ∧ ?f' y = x)
    proof−
      have ∀x::real. (x≥0 ⟶ (∃y. (y ∈ norms ∧ f y = x )))
          by (metis (no-types, opaque-lifting) bij-betw-iff-bijections mem-Collect-eq
normed-gyrolinear-space-axioms normed-gyrolinear-space-def norms-def)
      moreover have ∀x::real. (x<0 ⟶ (∃y. (y ∈ norms ∧ (f y) = −x)))
        by (simp add: calculation)
      moreover have ∀x::real. (x≥0 ⟶ (∃y. (y ∈ norms ∧ ?f' y = x )))
          by (smt (z3) calculation(1) f-inv-into-f normed-gyrolinear-space-axioms
normed-gyrolinear-space-def norms-def)
       moreover have ∀x::real. (x<0 ⟶ (∃y. (y ∈ norms-neg ∧ (f (−y)) =
−x)))
        using calculation(2) norms-def norms-neg-def by auto
      moreover have ∀x::real. (x<0 ⟶ (∃y. (y ∈ norms-neg ∧ (?f' (−y)) =
−x)))
     by (smt (z3) calculation(1) calculation(4) f-inv-into-f normed-gyrolinear-space-axioms
normed-gyrolinear-space-def norms-def norms-neg-def rangeI)
      moreover have ∀x::real. (x≥ 0 ∨ x<0)
        by (simp add: linorder-le-less-linear)
      ultimately show ?thesis
      proof −
        { fix rr :: real
          have ff1: ∀r. (r::real) < 0 ∨ 0 ≤ r
            by (smt (z3))
          have ff2: ∀r ra. sup (ra::real) r = sup r ra
            by (smt (z3) inf-sup-aci(5))
          have ff3: ∀R Ra. (Ra::real set) ∪ R = R ∪ Ra
            by (smt (z3) Un-commute)
          have ff4: ∀r ra. (r::real) ≤ sup r ra
            by simp
          have ff5: ∀R Ra. (R::real set) ⊆ Ra ∪ R
            by (smt (z3) inf-sup-ord(4))
          have ff6: ∀r. (r::real) ≤ r
            by (smt (z3))
```

have *ff7*: $\forall r\ R\ Ra.\ (r{::}real) \notin R \lor r \in Ra \lor \neg\ R \subseteq Ra$
by *blast*
have *ff8*: $\forall r.\ -\ (-\ (r{::}real)) = r$
using *verit-minus-simplify*(*4*) **by** *blast*
have *ff9*: $-\ (0{::}real) = 0$
by (*smt* (*z3*))
have $\forall r\ ra.\ r \notin$ *norms-all* $\lor$ (*if* $r = 0$ *then 0 else if* $r \in$ *norms then f r else if* $r \in$ *norms-neg then* $-\ f\ (-\ r)$ *else undefined*) $\neq$ (*if* $ra = 0$ *then 0 else if* $ra \in$ *norms then f ra else if* $ra \in$ *norms-neg then* $-\ f\ (-\ ra)$ *else undefined*) $\lor\ ra \notin$ *norms-all* $\lor\ r = ra$
using ‹$\forall x\ y.\ x \in$ *norms-all* $\land\ y \in$ *norms-all* $\land$ (*if* $x = 0$ *then 0 else if* $x \in$ *norms then f x else if* $x \in$ *norms-neg then* $-\ f\ (-\ x)$ *else undefined*) $=$ (*if* $y = 0$ *then 0 else if* $y \in$ *norms then f y else if* $y \in$ *norms-neg then* $-\ f\ (-\ y)$ *else undefined*) $\longrightarrow x = y$› **by** *blast*
then have $\forall r.$ (*if* $r = 0$ *then 0 else if* $r \in$ *norms then f r else if* $r \in$ *norms-neg then* $-\ f\ (-\ r)$ *else undefined*) $\neq$ (*if True then 0 else if* $0 \in$ *norms then f 0 else if* $0 \in$ *norms-neg then* $-\ f\ 0$ *else undefined*) $\lor\ r = 0 \lor 0 \notin$ *norms-all* $\lor\ r \notin$ *norms-all*
using *ff9* **by** (*smt* (*z3*))
then have ($\exists r.$ (*if* $r = 0$ *then 0 else if* $r \in$ *norms then f r else if* $r \in$ *norms-neg then* $-\ f\ (-\ r)$ *else undefined*) $= rr \land r \in$ *norms-all*) $\lor$ ($\exists r.$ (*if* $r = 0$ *then 0 else if* $r \in$ *norms then f r else if* $r \in$ *norms-neg then* $-\ f\ (-\ r)$ *else undefined*) $= rr \land r \in$ *norms-all*)
using *ff9 ff8 ff7 ff6 ff5 ff4 ff3 ff2 ff1* ‹$\forall x{<}0.\ \exists y.\ y \in$ *norms-neg* $\land\ f\ (-\ y) = -\ x$› ‹$\forall x{\geq}0.\ \exists y.\ y \in$ *norms* $\land$ (*if* $y = 0$ *then 0 else if* $y \in$ *norms then f y else if* $y \in$ *norms-neg then* $-\ f\ (-\ y)$ *else undefined*) $= x$› ‹$\forall x{\geq}0.\ \exists y.\ y \in$ *norms* $\land\ f\ y = x$› *if-True norms-all-def zero-only-norms-norms-neg* **by** *moura* **}**
then show *?thesis*
by *blast*
qed

qed
moreover have *inj-on ?f′ norms-all*
using $*$ *inj-on-def* **by** *blast*
moreover have $***$:$\forall x{::}real.\ \exists y{\in}$*norms-all*. (*?f′* $y = x$)
using $**$ **by** *blast*
moreover have *?f′* ‘ *norms-all* = *UNIV*
proof−
have *?f′* ‘ *norms-all* $\subseteq$ *UNIV*
by *blast*
moreover have *UNIV* $\subseteq$ *?f′* ‘ *norms-all*
proof−
fix $x{::}real$
have $\exists y{\in}$*norms-all*. (*?f′* $y = x$)
using $**$ **by** *blast*
then have $x \in$ (*?f′* ‘ *norms-all*)
by *blast*
then have $\forall x{::}real.$ ($x \in$ (*?f′* ‘ *norms-all*))
by (*smt* (*verit*, *del-insts*) $**$ *image-iff*)

then show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *force*
qed
ultimately show *bij-betw ?f′ norms-all UNIV*
using *bij-betw-def* **by** *blast*
qed

moreover have *fact-fin*: $((\forall x::real.\ \forall y::real.\ (\ x \in norms\text{-}all \wedge y \in norms\text{-}all \wedge x > y) \longrightarrow (?f'\ x) > (?f'\ y))$
$\wedge\ (?f'\ 0) = 0 \wedge bij\text{-}betw\ ?f'\ norms\text{-}all\ UNIV)$
using *fact1 fact2* **by** *argo*

ultimately show *?thesis*
using *fact-fin*
by (*smt* (*verit*, *del-insts*))
qed

end

locale *normed-gyrolinear-space′* =
fixes $norm'::'a::gyrolinear\text{-}space \Rightarrow real$
fixes $f'::real \Rightarrow real$
assumes $\forall a::'a.\ (norm'\ a \geq 0)$
assumes $bij\text{-}betw\ f'\ ((norm'\ `\ UNIV) \cup ((\lambda x.\ -1 * norm'\ x)\ `\ UNIV))\ UNIV$
assumes $\forall y::real.\ \forall z::real.\ ((\ y \in\ ((norm'\ `\ UNIV) \cup ((\lambda x.\ -1 * norm'\ x)\ `\ UNIV)) \wedge$
$z \in\ ((norm'\ `\ UNIV) \cup ((\lambda x.\ -1 * norm'\ x)\ `\ UNIV)) \wedge y > z) \longrightarrow (f'\ y) > (f'\ z))$
assumes $f'\ 0 = 0$
assumes $\forall x::'a.\ \forall y::'a.\ f'(norm'\ (gyroplus\ x\ y)) \leq (f'\ (norm'\ x)) + (f'\ (norm'\ y))$
assumes $f'\ (norm'\ (scale\ r\ x)) = |r| * (f'\ (norm'\ x))$
assumes $norm'\ (gyr\ u\ v\ x) = norm'\ x$
assumes $\forall x::'a.\ ((norm'\ x) = 0 \longleftrightarrow x = gyrozero)$
begin

definition *norms::real set* **where**
$norms = norm'\ `\ UNIV$

definition *norms-neg::real set* **where**
$norms\text{-}neg = (\lambda x.\ -1 * norm'\ x)\ `\ UNIV$

definition *norms-all::real set* **where**
$norms\text{-}all = norms \cup norms\text{-}neg$

lemma *norms-neq-not-empty*:

shows *norms-neg* $\neq$ {}
using *add.inverse-inverse norms-neg-def* **by** *fastforce*

lemma *zero-only-norms-norms-neg*:
assumes *x*$\in$*norms x*$\in$*norms-neg*
shows *x=0*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*1*) *assms*(*2*) *f-inv-into-f normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-def norms-neg-def*)

definition *norm-oplus-f*::*real* $\Rightarrow$ *real* $\Rightarrow$ *real* (**infixl** $\oplus_f$ *105*)
where $a \oplus_f b$ = (*if* (*a*$\in$*norms-all* $\wedge$ *b*$\in$*norms-all*) *then* (*inv-into norms-all f′*) ((*f′ a*) + (*f′ b*))
else undefined)

definition *norm-otimes-f*::*real* $\Rightarrow$ *real* $\Rightarrow$ *real* (**infixl** $\otimes_f$ *105*)
where $r \otimes_f a$ = (*if* (*a*$\in$*norms-all*) *then* (*inv-into norms-all f′*) (*r* $*$ (*f′ a*))
else undefined)

lemma *vector-space-of-norms*:
shows *vector-space-with-domain norms-all norm-oplus-f 0 norm-otimes-f*
proof
fix *x y*
show *x* $\in$ *norms-all* $\Longrightarrow$ *y* $\in$ *norms-all* $\Longrightarrow$ $x \oplus_f y$ $\in$ *norms-all*
proof−
assume *x*$\in$*norms-all*
show *y* $\in$ *norms-all* $\Longrightarrow$ $x \oplus_f y$ $\in$ *norms-all*
proof−
assume *y*$\in$*norms-all*
show $x \oplus_f y$ $\in$ *norms-all*
by (*smt* (*verit*, *del-insts*) *UNIV-I* ‹*x* $\in$ *norms-all*› ‹*y* $\in$ *norms-all*› *bij-betw-def inv-into-into norm-oplus-f-def normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-all-def norms-def*)
qed
qed
next
show *0* $\in$ *norms-all*
by (*metis Un-iff normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-all-def norms-def rangeI*)
next
fix *x y z*
show *x* $\in$ *norms-all* $\Longrightarrow$
y $\in$ *norms-all* $\Longrightarrow$ *z* $\in$ *norms-all* $\Longrightarrow$ $x \oplus_f y \oplus_f z = x \oplus_f (y \oplus_f z)$
proof−
assume *x*$\in$*norms-all*
show *y* $\in$ *norms-all* $\Longrightarrow$ *z* $\in$ *norms-all* $\Longrightarrow$ $x \oplus_f y \oplus_f z = x \oplus_f (y \oplus_f z)$
proof−

```
      assume y ∈ norms-all
      show z ∈ norms-all ⟹ x ⊕f y ⊕f z = x ⊕f (y ⊕f z)
      proof-
        assume z ∈ norms-all
        show x ⊕f y ⊕f z = x ⊕f (y ⊕f z)
        proof-
          have x ⊕f y = (inv-into norms-all f') ((f' x) + (f' y))
            by (simp add: ‹x ∈ norms-all› ‹y ∈ norms-all› norm-oplus-f-def)
          moreover have x ⊕f y ⊕f z = (inv-into norms-all f') ((
        f' ( (inv-into norms-all f') ((f' x) + (f' y)))) + (f' z))
          by (metis (no-types, lifting) UNIV-I ‹z ∈ norms-all› bij-betw-imp-surj-on
calculation inv-into-into norm-oplus-f-def normed-gyrolinear-space'.norms-neg-def
normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def)
            moreover have x ⊕f y ⊕f z = (inv-into norms-all f') (((f' x)+ (f'
y))+(f' z))
          by (metis (mono-tags, lifting) UNIV-I bij-betw-imp-surj-on calculation(2)
f-inv-into-f normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms
normed-gyrolinear-space'-def norms-all-def norms-def)
          moreover have (y ⊕f z) = (inv-into norms-all f') ((f' y) + (f' z))
            by (simp add: ‹y ∈ norms-all› ‹z ∈ norms-all› norm-oplus-f-def)
          moreover have x ⊕f (y ⊕f z) = (inv-into norms-all f') ((f' x) +
        (f' ((inv-into norms-all f') ((f' y) + (f' z)))))
            by (metis (mono-tags, lifting) UNIV-I ‹x ∈ norms-all› bij-betw-def cal-
culation(4) inv-into-into norm-oplus-f-def normed-gyrolinear-space'.norms-neg-def
normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def)
          moreover have x ⊕f (y ⊕f z) = (inv-into norms-all f') ((f' x) +
        ((f' y) + (f' z)))
          by (metis (mono-tags, lifting) UNIV-I bij-betw-imp-surj-on calculation(5)
f-inv-into-f normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms
normed-gyrolinear-space'-def norms-all-def norms-def)
          ultimately show ?thesis
            by argo
        qed
      qed
    qed
  qed
next
  fix x y
  show x ∈ norms-all ⟹ y ∈ norms-all ⟹ x ⊕f y = y ⊕f x
  proof-
    assume x∈ norms-all
    show y ∈ norms-all ⟹ x ⊕f y = y ⊕f x
    proof-
      assume y ∈ norms-all
      show x ⊕f y = y ⊕f x
        by (simp add: add.commute norm-oplus-f-def)
    qed
  qed
next
```

```
  fix x
  show  x ∈ norms-all ⟹ x ⊕f 0 = x
  proof−
    assume x∈norms-all
    show x ⊕f 0 = x
    proof−
      have x ⊕f 0 = (inv-into norms-all f') ((f' x) + (f' 0))
           by (metis (mono-tags, lifting) Un-iff ‹x ∈ norms-all› norm-oplus-f-def
normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def
rangeI)
      then show ?thesis
      by (smt (verit, del-insts) ‹x ∈ norms-all› bij-betw-def inv-into-f-eq normed-gyrolinear-space'.norms-neg-def
normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def)
    qed
  qed
next
  fix x
  show x ∈ norms-all ⟹ ∃ y∈norms-all. x ⊕f y = 0
  proof−
    assume x∈norms-all
    show  ∃ y∈norms-all. x ⊕f y = 0
    proof−
      let ?y = (inv-into norms-all f') (−(f' x))
      have  x ⊕f ?y = (inv-into norms-all f') ((f' x) + (f' ?y))
            by (smt (verit, ccfv-SIG) ‹x ∈ norms-all› bij-betwE bij-betw-inv-into
f-inv-into-f inv-into-into norm-oplus-f-def normed-gyrolinear-space'.norms-neg-def
normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def
rangeI)
      moreover have  x ⊕f ?y = (inv-into norms-all f') ((f' x) + (−(f' x)))
       by (smt (verit, ccfv-SIG) bij-betw-inv-into-right calculation iso-tuple-UNIV-I
normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def
norms-all-def norms-def)
      moreover have x ⊕f ?y =(inv-into norms-all f') 0
        using calculation(2) by force
      moreover have x ⊕f ?y = 0
         by (metis (no-types, lifting) Un-iff bij-betw-def calculation(3) inv-into-f-eq
normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def
norms-all-def norms-def rangeI)
      moreover have ?y ∈ norms-all
           by (metis (no-types, lifting) UNIV-I bij-betw-imp-surj-on inv-into-into
normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def
norms-all-def norms-def)
      ultimately show ?thesis
        by blast
    qed
  qed
next
  fix x a
  show x ∈ norms-all ⟹ a ⊗f x ∈ norms-all
```

proof−
assume $x{\in}$*norms-all*
show $a \otimes_f x \in$ *norms-all*
by (*smt* (*verit*, *best*) ‹$x \in$ *norms-all*› *bij-betw-imp-surj-on bij-betw-inv-into norm-otimes-f-def normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-all-def norms-def rangeI*)
qed
next
fix $x\ a\ b$
show $x \in$ *norms-all* $\Longrightarrow (a + b) \otimes_f x = a \otimes_f x \oplus_f (b \otimes_f x)$
proof−
assume $x{\in}$*norms-all*
show $(a + b) \otimes_f x = a \otimes_f x \oplus_f (b \otimes_f x)$
proof−
have $(a + b) \otimes_f x =$ (*inv-into norms-all* f') $((a{+}b) * (f'\ x))$
using ‹$x \in$ *norms-all*› *norm-otimes-f-def* **by** *presburger*
moreover have $(a + b) \otimes_f x =$ (*inv-into norms-all* f') $(a{*}(f'\ x) + b{*}(f'\ x))$
using *calculation* **by** *argo*
moreover have $*$: $a \otimes_f x \oplus_f (b \otimes_f x) =$ (*inv-into norms-all* f')
$((f'\ (a \otimes_f x)) + (f'\ (b \otimes_f x)))$
proof −
have $\bigwedge f.\ \neg$ *normed-gyrolinear-space′ norm′* $f\ \vee$ *bij-betw* f *norms-all UNIV*
by (*metis* (*no-types*) *normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-def norms-all-def norms-def*)
then show *?thesis*
by (*metis* (*full-types*) *UNIV-I* ‹$x \in$ *norms-all*› *bij-betw-imp-surj-on inv-into-into norm-oplus-f-def norm-otimes-f-def normed-gyrolinear-space′-axioms*)
qed
moreover have $**$: (*inv-into norms-all* f')
$((f'\ (a \otimes_f x)) + (f'\ (b \otimes_f x))) =$ (*inv-into norms-all* f')
$((f'$ ((*inv-into norms-all* f') $(a{*}(f'\ x)))) +$
$(f'$ ((*inv-into norms-all* f') $(b{*}(f'\ x)))))$
using ‹$x \in$ *norms-all*› *norm-otimes-f-def* **by** *presburger*
moreover have $a \otimes_f x \oplus_f (b \otimes_f x) =$ (*inv-into norms-all* f') $((a{*}(f'\ x)) + (b{*}(f'\ x)))$
using $*$ $**$
by (*smt* (*verit*, *ccfv-threshold*) *UNIV-I bij-betw-imp-surj-on f-inv-into-f normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-all-def norms-def*)
ultimately show *?thesis*
by *presburger*
qed
qed
next
fix $x\ a\ b$
show $x \in$ *norms-all* $\Longrightarrow a \otimes_f (b \otimes_f x) = (a * b) \otimes_f x$
proof−
assume $x{\in}$*norms-all*
show $a \otimes_f (b \otimes_f x) = (a * b) \otimes_f x$

by (*smt* (*verit*, *best*) *UNIV-I* ‹$x \in$ *norms-all*› *ab-semigroup-mult-class.mult-ac*(*1*) *bij-betw-imp-surj-on f-inv-into-f inv-into-into norm-otimes-f-def normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def*)
qed
next
fix x
show $x \in$ *norms-all* $\Longrightarrow$ *1* $\otimes_f\ x = x$
proof−
assume $x\in$*norms-all*
show *1* $\otimes_f\ x = x$
proof−
have *1* $\otimes_f\ x =$ (*inv-into norms-all* f') (*1*$*$(f' x))
using ‹$x \in$ *norms-all*› *norm-otimes-f-def* **by** *presburger*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) ‹$x \in$ *norms-all*› *bij-betw-inv-into-left lambda-one normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def*)
qed
qed
next
show $\bigwedge x\ y\ a.$
$x \in$ *norms-all* $\Longrightarrow$
$y \in$ *norms-all* $\Longrightarrow a \otimes_f (x \oplus_f y) = a \otimes_f x \oplus_f (a \otimes_f y)$
proof−
{
fix $x\ y\ a$
assume $x\in$ *norms-all* $\wedge$ $y\in$ *norms-all*
have $a \otimes_f (x \oplus_f y) =$ (*inv-into norms-all* f') ($a * f'$ ((*inv-into norms-all* f') ((f' x) + (f' y))))
by (*smt* (*verit*, *best*) *UNIV-I* ‹$x \in$ *norms-all* $\wedge$ $y \in$ *norms-all*› *bij-betw-imp-surj-on inv-into-into norm-oplus-f-def norm-otimes-f-def normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def*)
moreover have $a \otimes_f x \oplus_f (a \otimes_f y) =$ (*inv-into norms-all* f') ((f' (*inv-into norms-all* f' ($a *$ (f' x))))+(f' (*inv-into norms-all* f' ($a *$ (f' y)))))
by (*smt* (*verit*) ‹$x \in$ *norms-all* $\wedge$ $y \in$ *norms-all*› *bij-betw-def inv-into-into iso-tuple-UNIV-I normed-gyrolinear-space'.norm-oplus-f-def normed-gyrolinear-space'.norm-otimes-f-def normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def*)
ultimately have $a \otimes_f (x \oplus_f y) = a \otimes_f x \oplus_f (a \otimes_f y)$
using *UNIV-I bij-betw-imp-surj-on f-inv-into-f normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def norms-all-def norms-def norms-neg-def ring-class.ring-distribs*(*1*)
by (*smt* (*verit*, *best*) *normed-gyrolinear-space'.norms-neg-def*)
}
show $\bigwedge x\ y\ a.$
$x \in$ *norms-all* $\Longrightarrow$
$y \in$ *norms-all* $\Longrightarrow a \otimes_f (x \oplus_f y) = a \otimes_f x \oplus_f (a \otimes_f y)$
using ‹$\bigwedge y\ x\ a.\ x \in$ *norms-all* $\wedge$ $y \in$ *norms-all* $\Longrightarrow a \otimes_f (x \oplus_f y) = a \otimes_f x \oplus_f (a \otimes_f y)$› **by** *blast*
qed

qed

lemma *r2*:
shows $norm'\ (x \oplus y) \leq (norm'\ x)\ \oplus_f\ (norm'\ y)$
proof−
have $(f'\ (norm'\ (x \oplus y))) \leq (f'\ (norm'\ x)) + (f'\ (norm'\ y))$
using *normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def* **by** *blast*
moreover have $(inv\text{-}into\ norms\text{-}all\ f'\ (f'\ (norm'\ (x \oplus y)))) \leq$
$(inv\text{-}into\ norms\text{-}all\ f'\ ((f'\ (norm'\ x)) + (f'\ (norm'\ y))))$
by (*smt* (*verit*, *ccfv-SIG*) *UNIV-I bij-betw-def f-inv-into-f inv-into-into normed-gyrolinear-space′.norms-neg-*
normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def norms-all-def norms-def)
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *UnI1 bij-betw-def inv-into-f-eq normed-gyrolinear-space′.norm-oplus-f-def*
normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def
norms-all-def norms-def rangeI)
qed

lemma *r3*:
shows $norm'\ (r\ \otimes\ x) =\ |r| \otimes_f\ (norm'\ x)$
by (*smt* (*verit*, *best*) *bij-betw-inv-into-left in-mono inf-sup-ord*(*3*) *norm-otimes-f-def*
normed-gyrolinear-space′.norms-neg-def normed-gyrolinear-space′-axioms normed-gyrolinear-space′-def
norms-all-def norms-def rangeI)

lemma *one-dim-vs*:
shows *one-dim-vector-space-with-domain norms-all norm-oplus-f 0 norm-otimes-f*
proof−
have *step1*: *vector-space-with-domain norms-all norm-oplus-f 0 norm-otimes-f*
using *vector-space-of-norms* **by** *auto*
moreover have *step2*: $\forall\, y.\ \forall\, x.\ (y \in\ norms\text{-}all\ \wedge$
$x \in\ norms\text{-}all \wedge x \neq 0\ \longrightarrow (\exists!r{::}real.\ y = r \otimes_f\ x))$
proof
fix y
show $\forall\, x.\ (y \in\ norms\text{-}all\ \wedge$
$x \in\ norms\text{-}all \wedge x \neq 0\ \longrightarrow (\exists!r{::}real.\ y = r \otimes_f\ x))$
proof
fix x
show $y \in\ norms\text{-}all\ \wedge$
$x \in\ norms\text{-}all \wedge x \neq 0\ \longrightarrow (\exists!r{::}real.\ y = r \otimes_f\ x)$
proof
assume $y \in\ norms\text{-}all\ \wedge$
$x \in\ norms\text{-}all \wedge x \neq 0$
show $(\exists!r{::}real.\ y = r \otimes_f\ x)$
proof−
have $(\exists\, r{::}real.\ y = r \otimes_f\ x)$
proof−
let $?r = f'(y)/f'(x)$
have $?r \otimes_f\ x = (inv\text{-}into\ norms\text{-}all\ f')\ (?r * (f'\ x))$
by (*simp add*: ‹$y \in norms\text{-}all \wedge x \in norms\text{-}all \wedge x \neq 0$› *norm-otimes-f-def*)

```
          then show ?thesis
            by (smt (verit, ccfv-SIG) ‹y ∈ norms-all ∧ x ∈ norms-all ∧ x ≠ 0›
bij-betw-inv-into-left nonzero-eq-divide-eq normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axi
normed-gyrolinear-space'-def norms-all-def norms-def vector-space-of-norms vector-space-with-domain.zero-in-d
        qed

        moreover have ∀ r1.∀ r2. (y = r1 ⊗_f x ∧ y = r2 ⊗_f x ⟶ r1=r2)
        proof
          fix r1
          show ∀ r2. y = r1 ⊗_f x ∧ y = r2 ⊗_f x ⟶ r1=r2
          proof
            fix r2
            show y = r1 ⊗_f x ∧ y = r2 ⊗_f x ⟶ r1=r2
            proof
              assume y = r1 ⊗_f x ∧ y = r2 ⊗_f x
              show r1=r2
              proof−
                have r1 ⊗_f x = (inv-into norms-all f') (r1 ∗ (f' x))
        by (simp add: ‹y ∈ norms-all ∧ x ∈ norms-all ∧ x ≠ 0› norm-otimes-f-def)
        moreover have r2 ⊗_f x = (inv-into norms-all f') (r2 ∗ (f' x))
          using ‹y ∈ norms-all ∧ x ∈ norms-all ∧ x ≠ 0› norm-otimes-f-def by
presburger
        moreover
          have (inv-into norms-all f') (r1 ∗ (f' x)) = (inv-into norms-all f') (r2
∗ (f' x))
            using ‹y = r1 ⊗_f x ∧ y = r2 ⊗_f x› calculation(1) calculation(2) by
fastforce
          moreover have f' ( (inv-into norms-all f') (r1 ∗ (f' x))) =
      f'( (inv-into norms-all f') (r2 ∗ (f' x)))
            using calculation by presburger
          moreover have r1∗ (f' x) = r2∗ (f' x)
         by (metis (mono-tags, lifting) UNIV-I bij-betw-imp-surj-on calculation(3)
inv-into-injective normed-gyrolinear-space'.norms-neg-def normed-gyrolinear-space'-axioms
normed-gyrolinear-space'-def norms-all-def norms-def)
          ultimately show ?thesis
         by (metis (no-types, opaque-lifting) ‹y ∈ norms-all ∧ x ∈ norms-all ∧ x ≠
0› mult-right-cancel norm-oplus-f-def normed-gyrolinear-space'-axioms normed-gyrolinear-space'-def
vector-space-of-norms vector-space-with-domain.add-zero vector-space-with-domain.zero-in-dom)
        qed

      qed
    qed
  qed
  ultimately show ?thesis
    by blast
 qed
qed
qed
qed
```

ultimately show *?thesis*
by (*simp add*: *one-dim-vector-space-with-domain.intro one-dim-vector-space-with-domain-axioms.intro*)
qed

end

locale *normed-gyrolinear-space''* =
fixes *norm'*::*'a*::*gyrolinear-space* ⇒ *real*
fixes *oplus'*::*real* ⇒ *real* ⇒ *real*
fixes *otimes'*::*real*⇒*real* ⇒ *real*
assumes ∀ *a*::*'a*. (*norm' a* ≥ *0*)
assumes *ax-space*: *one-dim-vector-space-with-domain* ((*norm'* ‘ *UNIV*) ∪ ((λ*x*. −*1* ∗ *norm' x*) ‘ *UNIV*))
oplus' 0 otimes'
assumes *ax3*: ∀ *x*::*'a*. ∀ *y*::*'a*. (*norm'* (*gyroplus x y*)) ≤ *oplus'* (*norm' x*) (*norm' y*)
assumes (*norm'* (*scale r x*)) = *otimes'* |*r*| (*norm' x*)
assumes *norm'* (*gyr u v x*) = *norm' x*
assumes ∀ *x*::*'a*. ((*norm' x*) = *0* ⟷ *x* = *gyrozero*)
begin

definition *norms*::*real set* **where**
norms = *norm'* ‘ *UNIV*

definition *norms-neg*::*real set* **where**
norms-neg = (λ*x*. −*1* ∗ *norm' x*) ‘ *UNIV*

definition *norms-all*::*real set* **where**
norms-all = *norms* ∪ *norms-neg*

lemma *norms-neq-not-empty*:
shows *norms-neg* ≠ {}
using *add.inverse-inverse norms-neg-def* **by** *fastforce*

lemma *zero-only-norms-norms-neg*:
assumes *x*∈*norms x*∈*norms-neg*
shows *x*=*0*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*1*) *assms*(*2*) *f-inv-into-f normed-gyrolinear-space''-axioms normed-gyrolinear-space''-def norms-def norms-neg-def*)

lemma *not-trivial-domen-has-pos*:
assumes ∃ *x*. (*x*∈*norms-all* ∧ *x*≠*0*)
shows ∃ *x*. (*x*∈*norms* ∧ *x*≠*0*)
using *assms norms-all-def norms-def norms-neg-def* **by** *auto*

lemma *iso-with-real*:
assumes ∃ *x*. (*x*∈*norms-all* ∧ *x*≠*0*)
shows ∃ *g*. (*bij-betw g norms-all UNIV* ∧ (*g 0*) = *0* ∧

(∀ u.∀ v. (u∈norms-all ∧ v∈norms-all ⟶ g (oplus′ u v) = (g u) + (g v)))
∧ (∀ u.∀ r::real. (u∈norms-all ⟶ g (otimes′ r u) = r∗(g u)))
)
proof−
obtain *x* **where** *x∈norms ∧ x≠0*
using *assms not-trivial-domen-has-pos* **by** *presburger*
moreover have *x∈ norms-all*
by (*simp add*: *calculation norms-all-def*)
have ∀ *y*. (*y∈norms-all* ⟶ (∃!*r*.(*y* = *otimes′ r x*)))
using *ax-space one-dim-vector-space-with-domain-axioms-def*
by (*metis* ‹*x* ∈ *norms-all*› *calculation norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*2*))
let *?g* = λ*y*. (*THE r*. *y* = *otimes′ r x*)
have *bij-betw ?g norms-all UNIV*
proof−
have *inj-on ?g norms-all*
by (*smt* (*verit*, *best*) ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› *inj-on-def the-equality*)
moreover have ∀ *r*::*real*. ∃ *y*. (*y*∈ *norms-all* ∧ *y* = *otimes′ r x*)
by (*metis* ‹*x* ∈ *norms-all*› *ax-space norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *vector-space-with-domain.smult-closed*)
moreover have ∀ *r*::*real*.∃ *y*∈*norms-all*. *?g y* = *r*
using ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› *calculation*(*2*) **by** *blast*
ultimately show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *UNIV-eq-I bij-betw-apply inj-on-imp-bij-betw*)
qed
moreover have *?g 0* = *0*
proof−
obtain *r* **where** *0* = *otimes′ r x*
by (*metis* ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› *ax-space norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain-def vector-space-with-domain.zero-in-dom*)

moreover obtain *xx* **where** *x*=*norm′ xx*
using *norms-all-def*
using *norms-def norms-neg-def*
using ‹*x* ∈ *norms* ∧ *x* ≠ *0*› **by** *auto*

moreover have *otimes′ 0 x* = *norm′* (*0* ⊗ *xx*)
by (*metis* (*no-types*, *lifting*) *calculation*(*2*) *norm-zero normed-gyrolinear-space′′-axioms normed-gyrolinear-space′′-def real-norm-def*)
moreover have *otimes′ 0 x* = *0*
by (*smt* (*verit*, *ccfv-threshold*) ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› ‹*x* ∈ *norms-all*› *ax-space norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *vector-space-with-domain-def*)
ultimately show *?thesis*
by (*smt* (*verit*) ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› *ax-space norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *the1-equality vector-space-with-domain.zero-in-dom*)
qed

moreover have ∀ *u*.∀ *v*. (*u*∈*norms-all* ∧ *v*∈*norms-all* ⟶ *?g* (*oplus′ u v*) = (*?g u*) + (*?g v*))
proof
fix *u*
show ∀ *v*. (*u*∈*norms-all* ∧ *v*∈*norms-all* ⟶ *?g* (*oplus′ u v*) = (*?g u*) + (*?g v*))
proof
fix *v*
show *u*∈*norms-all* ∧ *v*∈*norms-all* ⟶ *?g* (*oplus′ u v*) = (*?g u*) + (*?g v*)
proof
assume *u*∈*norms-all* ∧ *v*∈*norms-all*
show *?g* (*oplus′ u v*) = (*?g u*) + (*?g v*)
proof−
obtain *a* **where** *u* = *otimes′ a x*
using ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› ‹*u* ∈ *norms-all* ∧ *v* ∈ *norms-all*› **by** *blast*
moreover obtain *b* **where** *v* = *otimes′ b x*
using ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› ‹*u* ∈ *norms-all* ∧ *v* ∈ *norms-all*› **by** *blast*
moreover have ∗:*oplus′ u v* = *otimes′* (*a*+*b*) *x*
by (*metis* ‹*x* ∈ *norms-all*› *ax-space calculation*(*1*) *calculation*(*2*) *norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain-def vector-space-with-domain.smult-distr-sadd*)
moreover have *oplus′ u v* ∈ *norms-all*
by (*metis* ∗ ‹*x* ∈ *norms-all*› *ax-space norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *vector-space-with-domain.smult-closed*)
moreover have *?g* (*oplus′ u v*) = (*a*+*b*)
using ∗
using ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› *calculation*(*4*) **by** *auto*
ultimately show *?thesis*
by (*smt* (*verit*, *del-insts*) ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› ‹*u* ∈ *norms-all* ∧ *v* ∈ *norms-all*› *the1-equality*)
qed
qed
qed
qed
moreover have (∀ *u*.∀ *r*::*real*. (*u*∈*norms-all* ⟶ *?g* (*otimes′ r u*) = *r*∗(*?g u*)))
proof
fix *u*
show ∀ *r*::*real*. (*u*∈*norms-all* ⟶ *?g* (*otimes′ r u*) = *r*∗(*?g u*))
proof
fix *r*
show *u*∈*norms-all* ⟶ *?g* (*otimes′ r u*) = *r*∗(*?g u*)
proof
assume *u*∈*norms-all*
show *?g* (*otimes′ r u*) = *r*∗(*?g u*)
proof−
obtain *a* **where** *u* = *otimes′ a x*
using ‹∀ *y*. *y* ∈ *norms-all* ⟶ (∃!*r*. *y* = *otimes′ r x*)› ‹*u* ∈ *norms-all*›

```
by blast
          moreover have otimes′ r u = otimes′ (r∗a) x
            by (metis ‹x ∈ norms-all› ax-space calculation norms-all-def norms-def
norms-neg-def one-dim-vector-space-with-domain.axioms(1) vector-space-with-domain.smult-assoc)
          moreover have otimes′ r u ∈ norms-all
          by (metis ‹u ∈ norms-all› ax-space norms-all-def norms-def norms-neg-def
one-dim-vector-space-with-domain.axioms(1) vector-space-with-domain.smult-closed)
          moreover have ?g (otimes′ r u) = (r∗a)
             using ‹∀ y. y ∈ norms-all ⟶ (∃!r. y = otimes′ r x)› calculation(2)
calculation(3) by auto
          ultimately show ?thesis
           by (smt (verit, ccfv-threshold) ‹∀ y. y ∈ norms-all ⟶ (∃!r. y = otimes′
r x)› ‹u ∈ norms-all› theI′)
        qed
      qed
    qed
  qed

  ultimately show ?thesis
    by blast
qed

definition g-iso::(real⇒real)⇒bool where
  g-iso g ⟷ (bij-betw g norms-all UNIV ∧ (g 0) = 0 ∧
  (∀ u.∀ v. (u∈norms-all ∧ v∈norms-all ⟶ g (oplus′ u v) = (g u) + (g v)))
 ∧ (∀ u.∀ r::real. (u∈norms-all ⟶ g (otimes′ r u) = r∗(g u))))

lemma iso-neg-with-real:
  assumes ∃ x. (x∈norms-all ∧ x≠0)
  shows g-iso g ⟶ g-iso (λx. −1 ∗ (g x))
proof
  assume g-iso g
  show  g-iso (λx. −1 ∗ (g x))
  proof−
    have bij-betw (λx. −1 ∗ (g x)) norms-all UNIV
    proof−
      have inj-on (λx. −1 ∗ (g x)) norms-all
      by (smt (verit, ccfv-threshold) ‹g-iso g› bij-betw-imp-inj-on g-iso-def inj-on-def)
      moreover have ∀ r::real.∃ y∈norms-all. ((λx. −1 ∗ (g x)) y = r)
       by (metis UNIV-I ‹g-iso g› bij-betw-iff-bijections g-iso-def minus-equation-iff
mult-cancel-right2 mult-minus-left)
      ultimately show ?thesis
        by (metis (mono-tags, lifting) UNIV-eq-I bij-betwE bij-betw-imageI)
    qed
    moreover have  (λx. −1 ∗ (g x)) 0 = 0
      using ‹g-iso g› g-iso-def by force
    moreover have (∀ u.∀ v. (u∈norms-all ∧ v∈norms-all ⟶  (λx. −1 ∗ (g x))
(oplus′ u v)
 = ( (λx. −1 ∗ (g x)) u) + ( (λx. −1 ∗ (g x)) v)))
```

using ‹*g-iso g*› *g-iso-def* **by** *auto*
moreover have (∀ *u*.∀ *r*::*real*. (*u*∈*norms-all* ⟶ (λ*x*. −*1* ∗ (*g x*)) (*otimes′ r u*) = *r*∗((λ*x*. −*1* ∗ (*g x*)) *u*)))
using ‹*g-iso g*› *g-iso-def* **by** *auto*
ultimately show *?thesis*
using *g-iso-def* **by** *presburger*
qed
qed

lemma *iso-with-real-positive-on-norms*:
assumes ∃ *x*. (*x*∈*norms-all* ∧ *x*≠*0*)
shows ∃ *g*. (*g-iso g* ∧ (∀ *x*.(*x*∈*norms* ⟶ (*g x*)≥*0*))
∧ *bij-betw* (λ*x*. *if x* ∈ *norms then* (*g x*) *else undefined*) *norms* {*r*::*real*. *r*≥*0*})
proof−
obtain *xx* **where** *xx*∈*norms* ∧ *xx*≠*0*
using *assms not-trivial-domen-has-pos* **by** *blast*
moreover obtain *x* **where** *norm′ x* = *xx*
using *calculation norms-def* **by** *auto*
moreover obtain *g* **where** *g-iso g*
using *iso-with-real*
using *assms g-iso-def* **by** *blast*
let *?g* = *if* (*g xx*) < *0 then* (λ*x*. −*1* ∗ (*g x*)) *else g*
have ∗:*?g xx* ≥ *0*
by *force*
moreover have *?g xx* ≠*0*
proof (*rule ccontr*)
assume ¬(*?g xx* ≠*0*)
have *?g xx* = *0*
using ‹¬ (*if g xx* < *0 then* λ*x*. − *1* ∗ *g x else g*) *xx* ≠ *0*› **by** *blast*
then have *?g xx* = *g xx*
by (*smt* (*verit*, *ccfv-threshold*))
then have *g xx* = *0*
by (*simp add*: ‹(*if g xx* < *0 then* λ*x*. − *1* ∗ *g x else g*) *xx* = *0*›)
then have *xx*=*0*
by (*metis* ‹*g-iso g*› *ax-space bij-betw-iff-bijections calculation*(*1*) *g-iso-def in-mono inf-sup-ord*(*3*) *norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *vector-space-with-domain.zero-in-dom*)
then show *False*
using *calculation*(*1*) **by** *blast*
qed
moreover have *g-iso ?g*
using ‹*g-iso g*› *assms iso-neg-with-real* **by** *presburger*
moreover have ∀ *x*.(*x*∈*norms* ⟶ (*?g x*)≥*0*)
proof(*rule ccontr*)
assume ¬(∀ *x*.(*x*∈*norms* ⟶ (*?g x*)≥*0*))
have ∃ *x*. (*x*∈*norms* ∧ (*?g x*) < *0*)
using ‹¬ (∀ *x*. *x* ∈ *norms* ⟶ *0* ≤ (*if g xx* < *0 then* λ*x*. − *1* ∗ *g x else g*) *x*)› **by** *fastforce*
moreover obtain *yy* **where** *yy* ∈ *norms* ∧ (*?g yy*) <*0*

```
      using calculation by blast
    moreover obtain y where norm′ y = yy
      using calculation(2) norms-def by auto
    let ?A = {norm′ (r ⊗ x) | r::real. True}
    let ?B = {norm′ (r ⊗ y) | r::real. True}
    have ?A ∪ ?B ⊆ norms
      using norms-def by auto
    let ?gA = {(?g a)|a. a∈?A}
    have ?gA = {r::real. r≥0}
    proof−
      have ∀ a. (a∈?A ⟶ ?g a ≥0)
      proof
        fix a
        show (a∈?A ⟶ ?g a ≥0)
        proof
            assume a∈?A
            show ?g a ≥0
            proof−
              obtain r where a = norm′ (r ⊗ x)
                using ‹a ∈ {norm′ (r ⊗ x) |r. True}› by blast
              moreover have ?g a = ?g (norm′ (r ⊗ x) )
                using calculation by presburger
              moreover have ?g a = ?g ( otimes′ |r| (norm′ x))
            by (metis calculation(1) normed-gyrolinear-space′′-axioms normed-gyrolinear-space′′-def)
              moreover have ?g a = |r| ∗ ?g (norm′ x)
                using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› ‹norm′ x = xx›
‹xx ∈ norms ∧ xx ≠ 0› calculation(3) g-iso-def norms-all-def by auto
              ultimately show ?thesis
                by (simp add: ‹norm′ x = xx›)
            qed
        qed
      qed
      moreover have ?gA ⊆ {r::real. r≥0}
        using calculation by fastforce
      moreover have {r::real. r≥0} ⊆ ?gA
      proof−
        have bij-betw ?g norms-all UNIV
          using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› g-iso-def by blast
        moreover have ∀ r::real. (r≥0 ⟶ r∈?gA)
        proof
          fix r
          show r≥0 ⟶ r∈?gA
          proof
            assume r≥0
            show r∈?gA
            proof−
              obtain r′ where |r′| = r / (?g xx)
                using ∗
                by (meson ‹0 ≤ r› abs-of-nonneg divide-nonneg-nonneg)
```

moreover have $r = |r'| * (?g\ xx)$
by (*simp add*: ‹(*if g xx* $<$ *0 then* $\lambda x.\ -\ 1 * g\ x$ *else g*) *xx* $\neq$ *0*›
calculation)
moreover have $r = |r'| * (?g\ (norm'\ x))$
using ‹$norm'\ x = xx$› *calculation*(*2*) **by** *blast*
moreover have $r = ?g\ (otimes'\ |r'|\ (norm'\ x))$
using ‹*g-iso* (*if g xx* $<$ *0 then* $\lambda x.\ -\ 1 * g\ x$ *else g*)› ‹$norm'\ x = xx$›
‹$xx \in norms \wedge xx \neq 0$› *calculation*(*3*) *g-iso-def norms-all-def* **by** *auto*
moreover have $r = ?g\ (norm'\ (|r'|\ \otimes x))$
by (*smt* (*verit*, *del-insts*) *calculation*(*4*) *normed-gyrolinear-space''-axioms*
normed-gyrolinear-space''-def)
ultimately show *?thesis*
by *blast*
qed
qed
qed
ultimately show *?thesis*
by *blast*
qed

ultimately show *?thesis*
by *fastforce*
qed
let $?gB = \{(?g\ b)|b.\ b\in ?B\}$
have $?gB = \{r{::}real.\ r\leq 0\}$

proof−
have $\forall\, a.\ (a\in ?B \longrightarrow ?g\ a \leq 0)$
proof
fix *a*
show $(a\in ?B \longrightarrow ?g\ a \leq 0)$
proof
assume $a\in ?B$
show $?g\ a\leq 0$
proof−
obtain *r* **where** $a = norm'\ (r \otimes y)$
using ‹$a \in \{norm'\ (r \otimes y)\ |r.\ True\}$› **by** *blast*
moreover have $?g\ a = ?g\ (norm'\ (r \otimes y)\ )$
using *calculation* **by** *presburger*
moreover have $?g\ a = ?g\ (\ otimes'\ |r|\ (norm'\ y))$
by (*metis calculation*(*1*) *normed-gyrolinear-space''-axioms normed-gyrolinear-space''-def*)
moreover have $?g\ a = |r| * ?g\ (norm'\ y)$
using ‹*g-iso* (*if g xx* $<$ *0 then* $\lambda x.\ -\ 1 * g\ x$ *else g*)› ‹$norm'\ y =$
yy› ‹$yy \in norms \wedge$ (*if g xx* $<$ *0 then* $\lambda x.\ -\ 1 * g\ x$ *else g*) *yy* $<$ *0*› *calculation*(*3*)
g-iso-def norms-all-def **by** *auto*

ultimately show *?thesis*
by (*simp add*: ‹$norm'\ y = yy$› ‹$yy \in norms \wedge$ (*if g xx* $<$ *0 then* $\lambda x.$

```
− 1 ∗ g x else g) yy < 0› mult-le-0-iff order-less-imp-le)
          qed
        qed
      qed
      moreover have ?gB ⊆ {r::real. r≤0}
        using calculation by fastforce
      moreover have {r::real. r≤0} ⊆ ?gB
      proof−
        have bij-betw ?g norms-all UNIV
          using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› g-iso-def by blast
        moreover have ∀ r::real. (r≤0 ⟶ r∈?gB)
        proof
          fix r
          show r≤0 ⟶ r∈?gB
          proof
            assume r≤0
            show r∈?gB
            proof−
              obtain r′ where |r′| = r / (?g yy)
                using ∗
                by (metis ‹r ≤ 0› ‹yy ∈ norms ∧ (if g xx < 0 then λx. − 1 ∗ g x
else g) yy < 0› abs-if divide-less-0-iff less-eq-real-def not-less-iff-gr-or-eq)
              moreover have r = |r′| ∗ (?g yy)
                using ‹yy ∈ norms ∧ (if g xx < 0 then λx. − 1 ∗ g x else g) yy < 0›
calculation by auto
              moreover have r = |r′| ∗ (?g (norm′ y))
                using ‹norm′ y = yy› calculation(2) by blast
              moreover have r = ?g (otimes′ |r′| (norm′ y))
                  using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› ‹norm′ y =
yy› ‹yy ∈ norms ∧ (if g xx < 0 then λx. − 1 ∗ g x else g) yy < 0› calculation(3)
g-iso-def norms-all-def by auto
              moreover have r = ?g (norm′ (|r′| ⊗ y))
            by (smt (verit, del-insts) calculation(4) normed-gyrolinear-space′′-axioms
normed-gyrolinear-space′′-def)
              ultimately show ?thesis
                by blast
            qed
          qed
        qed
        ultimately show ?thesis
          by blast
      qed

      ultimately show ?thesis
        by fastforce
    qed

    let ?gX-norms = {(?g x)|x. x∈norms}
    let ?gX-norms-all = {(?g x)|x. x∈norms-all}
```

let *?gA-union-B* = $\{(?g\ x)|x.\ x\in\ ?A\cup ?B\}$
have *?gA-union-B* $\subseteq$ *?gX-norms*
using ‹$\{norm'\ (r \otimes x)\ |r.\ True\} \cup \{norm'\ (r \otimes y)\ |r.\ True\} \subseteq norms$› **by** *force*
moreover have *?gA-union-B* = *?gA* $\cup$ *?gB*
proof−
have *?gA-union-B* $\subseteq$ *?gA* $\cup$ *?gB*
by *blast*
moreover have *?gA* $\cup$ *?gB* $\subseteq$ *?gA-union-B*
by *blast*
ultimately show *?thesis*
by *force*
qed
moreover have *?gA-union-B* = *UNIV*
using ‹$\{(if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g)\ a\ |a.\ a \in \{norm'\ (r \otimes x)\ |r.\ True\}\} = \{r.\ 0 \leq r\}$› ‹$\{(if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g)\ b\ |b.\ b \in \{norm'\ (r \otimes y)\ |r.\ True\}\} = \{r.\ r \leq 0\}$› *calculation*(*4*) **by** *force*
moreover have *UNIV* $\subseteq$ *?gX-norms*
using *calculation*(*3*) *calculation*(*5*) **by** *argo*

obtain *a* **where** $a\in norms\text{-}all \wedge \neg a\in norms$
by (*metis* (*mono-tags*, *lifting*) *Un-iff add.inverse-inverse assms mult-minus1 norms-all-def norms-def norms-neg-def rangeE rangeI zero-only-norms-norms-neg*)
let *?a* = *?g a*
have *?a* $\in$ *?gX-norms-all*

using ‹$a \in norms\text{-}all \wedge a \notin norms$› **by** *blast*

moreover have $\neg$*?a*$\in$ *?gX-norms*
proof(*rule ccontr*)
assume $\neg(\neg$*?a*$\in$ *?gX-norms*)
have *?a*$\in$*?gX-norms*
using ‹$\neg\ (if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g)\ a \notin \{(if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g)\ x\ |x.\ x \in norms\}$› **by** *blast*
then obtain *b* **where** $b\in norms \wedge ?g\ b = ?a$
by *force*

then show *False* **using** ‹$a \in norms\text{-}all \wedge a \notin norms$› ‹*g-iso* ($if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g$)› *bij-betw-inv-into-left g-iso-def inf-sup-ord*(*3*) *norms-all-def subsetD*
by (*smt* (*verit*, *ccfv-threshold*) ‹*g-iso g*›)
qed
moreover have *False*

using ‹$UNIV \subseteq \{(if\ g\ xx < 0\ then\ \lambda x. - 1 * g\ x\ else\ g)\ x\ |x.\ x \in norms\}$› *calculation*(*7*) **by** *blast*

ultimately show *False*

```
      by auto
    qed


  moreover have  bij-betw (λx. if x ∈ norms then (?g x) else undefined) norms
{r::real. r≥0}
  proof−
    let ?f = (λx. if x ∈ norms then (?g x) else undefined)
     let ?A = {norm′ (r ⊗ x) | r::real. True}
     let ?gA = {(?g a)|a. a∈?A}
     have s1:?gA = {r::real. r≥0}
        proof−
      have ∀ a. (a∈?A ⟶ ?g a ≥0)
      proof
        fix a
        show (a∈?A ⟶ ?g a ≥0)
        proof
            assume a∈?A
            show ?g a ≥0
            proof−
              obtain r where a = norm′  (r ⊗ x)
                using ‹a ∈ {norm′ (r ⊗ x) |r. True}› by blast
              moreover have ?g a = ?g (norm′  (r ⊗ x) )
                using calculation by presburger
              moreover have ?g a = ?g ( otimes′ |r| (norm′ x))
           by (metis calculation(1) normed-gyrolinear-space′′-axioms normed-gyrolinear-space′′-def)
              moreover have ?g a =  |r| ∗ ?g (norm′ x)
                using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› ‹norm′ x = xx›
‹xx ∈ norms ∧ xx ≠ 0› calculation(3) g-iso-def norms-all-def by auto
              ultimately show ?thesis
                by (simp add: ‹norm′ x = xx›)
            qed
        qed
      qed
      moreover have ?gA ⊆ {r::real. r≥0}
        using calculation by fastforce
      moreover have {r::real. r≥0} ⊆ ?gA
      proof−
        have bij-betw ?g norms-all UNIV
          using ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› g-iso-def by blast
        moreover have ∀ r::real. (r≥0 ⟶ r∈?gA)
        proof
          fix r
          show r≥0 ⟶ r∈?gA
          proof
            assume r≥0
            show r∈?gA
            proof−
              obtain r′ where |r′| = r / (?g xx)
```

using $*$
by (*meson* ‹$0 \leq r$› *abs-of-nonneg divide-nonneg-nonneg*)
moreover have $r = |r'| * (?g\ xx)$
by (*simp add*: ‹(*if* $g\ xx < 0$ *then* $\lambda x. - 1 * g\ x$ *else* g) $xx \neq 0$› *calculation*)
moreover have $r = |r'| * (?g\ (norm'\ x))$
using ‹$norm'\ x = xx$› *calculation*(*2*) **by** *blast*
moreover have $r = ?g\ (otimes'\ |r'|\ (norm'\ x))$
using ‹*g-iso* (*if* $g\ xx < 0$ *then* $\lambda x. - 1 * g\ x$ *else* g)› ‹$norm'\ x = xx$› ‹$xx \in norms \wedge xx \neq 0$› *calculation*(*3*) *g-iso-def norms-all-def* **by** *auto*
moreover have $r = ?g\ (norm'\ (|r'| \otimes x))$
by (*smt* (*verit*, *del-insts*) *calculation*(*4*) *normed-gyrolinear-space''-axioms normed-gyrolinear-space''-def*)
ultimately show *?thesis*
by *blast*
qed
qed
qed
ultimately show *?thesis*
by *blast*
qed

ultimately show *?thesis*
by *fastforce*
qed
moreover have *s2*:$\forall y.\ (?g\ (norm'\ y) \geq 0)$
using ‹$\forall x.\ x \in norms \longrightarrow 0 \leq$ (*if* $g\ xx < 0$ *then* $\lambda x. - 1 * g\ x$ *else* g) x› *norms-def* **by** *blast*
moreover have $norms = ?A$
proof−
have $\forall y.\ (?g\ (norm'\ y) \in ?gA)$
using *s1 s2* **by** *blast*
moreover have $norms \subseteq ?A$
proof−
have $\forall y.\ (y \in norms \longrightarrow y \in ?A)$
proof
fix y
show $y \in norms \longrightarrow y \in ?A$
proof
assume $y \in norms$
show $y \in ?A$
proof−
obtain yy **where** $y = norm'\ yy$
using ‹$y \in norms$› *norms-def* **by** *auto*
moreover have $?g\ (norm'\ yy) \in ?gA$
using ‹$\forall y.$ (*if* $g\ xx < 0$ *then* $\lambda x. - 1 * g\ x$ *else* g) $(norm'\ y) \in \{$(*if* $g\ xx < 0$ *then* $\lambda x. - 1 * g\ x$ *else* g) $a\ |a.\ a \in \{norm'\ (r \otimes x)\ |r.\ True\}\}$› **by** *blast*
moreover have $norm'\ yy \in ?A$
proof−

```
obtain h where h ∈ ?A ∧ ?g h = ?g (norm′ yy)
  using calculation(2) by fastforce
moreover have ?g h ≥0
  using calculation s2 by blast

moreover {
  assume ?g = g
  have  g h = g (norm′ yy)
    by (smt (verit, ccfv-SIG) calculation(1))

  moreover have h=norm′ yy
  proof−
    have h∈norms
     using ‹h ∈ {norm′ (r ⊗ x) |r. True} ∧ (if g xx < 0 then λx. −
1 ∗ g x else g) h = (if g xx < 0 then λx. − 1 ∗ g x else g) (norm′ yy)› norms-def
by force
    moreover have norm′ yy ∈ norms
      using ‹y = norm′ yy› ‹y ∈ norms› by blast
    ultimately show ?thesis
       by (metis ‹g h = g (norm′ yy)› ‹g-iso g› bij-betw-inv-into-left
g-iso-def inf-sup-ord(3) norms-all-def subset-iff)
  qed
  ultimately have ?thesis
   using ‹h ∈ {norm′ (r ⊗ x) |r. True} ∧ (if g xx < 0 then λx. − 1
∗ g x else g) h = (if g xx < 0 then λx. − 1 ∗ g x else g) (norm′ yy)› by blast
}
  moreover {
  assume ?g = (λx. −1 ∗ (g x))
  have  g h = g (norm′ yy)
    by (smt (verit, ccfv-SIG) calculation(1))

  moreover have h=norm′ yy
  proof−
    have h∈norms
     using ‹h ∈ {norm′ (r ⊗ x) |r. True} ∧ (if g xx < 0 then λx. −
1 ∗ g x else g) h = (if g xx < 0 then λx. − 1 ∗ g x else g) (norm′ yy)› norms-def
by force
    moreover have norm′ yy ∈ norms
      using ‹y = norm′ yy› ‹y ∈ norms› by blast
    ultimately show ?thesis
       by (metis ‹g h = g (norm′ yy)› ‹g-iso g› bij-betw-inv-into-left
g-iso-def inf-sup-ord(3) norms-all-def subset-iff)
  qed
  ultimately have ?thesis
   using ‹h ∈ {norm′ (r ⊗ x) |r. True} ∧ (if g xx < 0 then λx. − 1
∗ g x else g) h = (if g xx < 0 then λx. − 1 ∗ g x else g) (norm′ yy)› by blast
}
ultimately show ?thesis
  by argo
```

```
            qed
            ultimately show ?thesis
              by fastforce
          qed
        qed
      qed
       show ?thesis
         using ‹∀ y. y ∈ norms ⟶ y ∈ {norm′ (r ⊗ x) |r. True}› by blast
     qed
     ultimately show ?thesis
       using norms-def by fastforce
   qed
    moreover have step1:inj-on ?f  norms
    proof−
      have ∀ x.∀ y. (x∈ norms ∧ y∈ norms ∧ (?f x) = (?f y) ⟶ x=y)
      proof
        fix x
        show ∀ y. (x∈ norms ∧ y∈ norms ∧ (?f x) = (?f y) ⟶ x=y)
        proof
          fix y
          show  (x∈ norms ∧ y∈ norms ∧ (?f x) = (?f y) ⟶ x=y)
         by (metis ‹g-iso (if g xx < 0 then λx. − 1 ∗ g x else g)› bij-betw-imp-inj-on
g-iso-def inf-sup-ord(3) inj-on-def norms-all-def subsetD)
        qed
      qed
      then show ?thesis
        using inj-on-def by blast
    qed
    moreover have ∀ r::real. (r≥0 ⟶ (∃ x. (x∈ norms ∧ ?f x = r)))
      by (smt (verit) calculation(3) mem-Collect-eq s1)

    moreover have step2:∀ r::real. (r≥0 ⟶ (∃ x∈ norms.( ?f x = r)))

      using calculation(5) by blast
    moreover have ∀ r∈{x::real. x≥0}. (∃ x∈norms. (?f x = r))
      using step2
      by blast
    moreover have ∗∗:?f=(λx. if x ∈ norms then (?g x) else undefined)
      by meson
    moreover have ?f ‘ norms = {r::real. r≥0}
      by (smt (verit) Collect-cong Setcompr-eq-image calculation(3) s1)
    ultimately show ?thesis
      by (simp add: bij-betw-def)

  qed

 ultimately show ?thesis
   by blast
qed
```

lemma *comparing-norms-help*:
  assumes $x \in norms$ $y \in norms\text{-}all$
  $x \leq y$
shows $y \in norms$
proof−
  have $x < y \vee x = y$
    using *assms*(*3*) **by** *argo*
  moreover {
    assume $x < y$
    have *?thesis*
      by (*smt* (*verit*) *Un-iff* ‹$x < y$› *add-0* *add-uminus-conv-diff* *assms*(*1*) *assms*(*2*) *full-SetCompr-eq* *linorder-not-less* *mem-Collect-eq* *mult-minus1* *normed-gyrolinear-space*′′*-axioms* *normed-gyrolinear-space*′′*-def* *norms-all-def* *norms-def* *norms-neg-def* *order-le-less-trans*)
  }
  moreover {
    assume $x = y$
    have *?thesis*
      using ‹$x = y$› *assms*(*1*) **by** *blast*
  }
  ultimately show *?thesis* **by** *blast*
qed

lemma *existence-of-f*:
 assumes $\exists x.\ (x \in norms\text{-}all \wedge x \neq 0)$
  shows $\exists f.\ (bij\text{-}betw\ f\ norms\ \{x{::}real.\ x \geq 0\}$
$\wedge\ (\forall y{::}real.\ \forall z{::}real.\ ((\ y \in\ norms \wedge$
$z \in\ norms \wedge y > z) \longrightarrow (f\ y) > (f\ z)))$
  $\wedge\ (\forall x.\ \forall y.\ f(norm'\ (x \oplus y)) \leq (f\ (norm'\ x)) + (f\ (norm'\ y)))$
$\wedge\ (\forall r{::}real.\ (\forall x.\ (f\ (norm'\ (r \otimes x)) = |r| * (f\ (norm'\ x))))))$
proof−
  obtain g **where** $(g\text{-}iso\ g \wedge (\forall x.(x \in norms \longrightarrow (g\ x) \geq 0))$
$\wedge\ bij\text{-}betw\ (\lambda x.\ if\ x \in\ norms\ then\ (g\ x)\ else\ undefined)\ norms\ \{r{::}real.\ r \geq 0\})$
    using *iso-with-real-positive-on-norms*
    assms **by** *blast*
  let $?f = \lambda x.\ if\ x \in\ norms\ then\ (g\ x)\ else\ undefined$
  have $\forall \alpha{::}real.\ \forall \beta{::}real.\ \forall x.\ ((0 \leq \alpha \wedge \alpha \leq \beta) \longrightarrow ((otimes'\ \alpha\ \ (norm'\ x)) \leq$
$(otimes'\ \beta\ \ (norm'\ x))))$
  proof
    fix α
    show $\forall \beta{::}real.\ \forall x.((0 \leq \alpha \wedge \alpha \leq \beta) \longrightarrow ((otimes'\ \alpha\ \ (norm'\ x)) \leq (otimes'$
$\beta\ \ (norm'\ x))))$
    proof
      fix β
     show $\forall x.((0 \leq \alpha \wedge \alpha \leq \beta) \longrightarrow ((otimes'\ \alpha\ \ (norm'\ x)) \leq (otimes'\ \beta\ \ (norm'$
$x))))$

```
      proof
        fix x
        show ((0 ≤ α ∧ α ≤ β) ⟶ ((otimes' α  (norm' x)) ≤ (otimes' β  (norm'
x))))
         proof
           assume 0 ≤ α ∧ α ≤ β
           show ((otimes' α  (norm' x)) ≤ (otimes' β  (norm' x)))
           proof−
             have otimes' α  (norm' x) = norm' (α ⊗ x)
           by (metis ‹0 ≤ α ∧ α ≤ β› abs-of-nonneg normed-gyrolinear-space''-axioms
normed-gyrolinear-space''-def)
             moreover have  norm' (α ⊗ x) = norm' ((((β+α)/2 − (β−α)/2)⊗ x)
               by (simp add: add-divide-distrib diff-divide-distrib)
             moreover have norm' (((β+α)/2 − (β−α)/2)⊗ x) =
             norm' (((β+α)/2) ⊗ x ⊕  (− (β−α)/2) ⊗ x )
           by (metis add.commute divide-minus-left scale-distrib uminus-add-conv-diff)
             moreover have  norm' (((β+α)/2) ⊗ x ⊕  (− (β−α)/2) ⊗ x )
         ≤  oplus' (norm' (((β+α)/2)⊗ x)) (norm' ((−(β−α)/2)  ⊗ x))
               using  ax3
               by blast
             moreover have −(β−α)/2 ≤0
               by (simp add: ‹0 ≤ α ∧ α ≤ β›)
             moreover have (β+α)/2 ≥0
               using ‹0 ≤ α ∧ α ≤ β› by auto
            moreover have ∗:(norm' (((β+α)/2)⊗ x)) =(otimes' ((β+α)/2) (norm'
x))
           by (smt (verit, ccfv-threshold) calculation(6) normed-gyrolinear-space''-axioms
normed-gyrolinear-space''-def)
             moreover have  |−(β−α)/2| = (β−α)/2
               using calculation(5) by force
                moreover have ∗∗:(norm' ((−(β−α)/2)⊗ x)) =(otimes' ((β−α)/2)
(norm' x))
           by (metis calculation(8) normed-gyrolinear-space''-axioms normed-gyrolinear-space''-def)
             moreover have   oplus' (norm' (((β+α)/2)⊗ x)) (norm' ((−(β−α)/2)
⊗ x)) =
        oplus' (otimes' ((β+α)/2) (norm' x)) (otimes' ( (β−α)/2) (norm' x))
               using ∗ ∗∗
               by presburger
          moreover have oplus' (otimes' ((β+α)/2) (norm' x)) (otimes' ( (β−α)/2)
(norm' x))
      = otimes' ((β+α)/2 + ((β−α)/2)) (norm' x)
              by (metis Un-iff ax-space one-dim-vector-space-with-domain-def rangeI
vector-space-with-domain.smult-distr-sadd)
             moreover have  otimes' ((β+α)/2 + ((β−α)/2)) (norm' x) = otimes'
β (norm' x)
               by argo
             ultimately show ?thesis
               by linarith
           qed
```

qed
qed
qed
qed
moreover have $\forall \alpha::real.\ \forall \beta::real.\ \forall x.\ ((0 < \alpha \wedge \alpha < \beta \wedge x \neq gyrozero) \longrightarrow ((otimes'\ \alpha\ \ (norm'\ x)) < (otimes'\ \beta\ \ (norm'\ x))))$
proof −
have $f1$: $\forall f\ fa\ fb.\ normed\text{-}gyrolinear\text{-}space''\ f\ fa\ fb = (((\forall a.\ 0 \leq f\ (a::'a)) \wedge one\text{-}dim\text{-}vector\text{-}space\text{-}with\text{-}domain\ (range\ f \cup range\ (\lambda a.\ -\ 1 * f\ a))\ fa\ 0\ fb \wedge (\forall a\ aa.\ f\ (a \oplus aa) \leq fa\ (f\ a)\ (f\ aa))) \wedge (\forall r\ a.\ f\ (r \otimes a) = fb\ (if\ r < 0\ then - r\ else\ r)\ (f\ a)) \wedge (\forall a\ aa\ ab.\ f\ (gyr\ a\ aa\ ab) = f\ ab) \wedge (\forall a.\ (f\ a = 0) = (a = 0_g)))$
by (*simp add*: *abs-if-raw normed-gyrolinear-space''-def*)
obtain $rr :: real \Rightarrow real$ **where**
$f2$: $bij\text{-}betw\ rr\ norms\text{-}all\ UNIV \wedge 0 = rr\ 0 \wedge (\forall r\ ra.\ r \in norms\text{-}all \wedge ra \in norms\text{-}all \longrightarrow rr\ (oplus'\ r\ ra) = rr\ r + rr\ ra) \wedge (\forall r\ ra.\ r \in norms\text{-}all \longrightarrow rr\ (otimes'\ ra\ r) = ra * rr\ r)$
using *assms iso-with-real* **by** *auto*
have $\forall a.\ (0 = norm'\ a) = (0_g = a)$
using *f1* **by** (*smt* (*z3*) *normed-gyrolinear-space''-axioms*)
then show *?thesis*
using *f2* **by** (*smt* (*z3*) *UnI2 bij-betw-inv-into-left calculation mult-right-cancel norms-all-def norms-def rangeI sup-commute*)
qed
moreover obtain *xx0* **where** $xx0 \in norms \wedge xx0 \neq 0$
using *assms not-trivial-domen-has-pos* **by** *blast*
moreover obtain *x0* **where** $xx0 = norm'\ x0$
using *calculation*(*3*) *norms-def* **by** *auto*
moreover have *mon*:$(\forall y\ z.\ y \in norms \wedge z \in norms \wedge z < y \longrightarrow ?f\ z < ?f\ y)$
proof
fix *y*
show $\forall z.\ (y \in norms \wedge z \in norms \wedge z < y \longrightarrow ?f\ z < ?f\ y)$
proof
fix *z*
show $y \in norms \wedge z \in norms \wedge z < y \longrightarrow ?f\ z < ?f\ y$
proof
assume $y \in norms \wedge z \in norms \wedge z < y$
show $?f\ z < ?f\ y$
proof−
let $?alpha = (?f\ y)/(?f\ (norm'\ x0))$
let $?beta = (?f\ z)/(?f\ (norm'\ x0))$
have $otimes'\ ?alpha\ (norm'\ x0) = y$
by (*smt* (*verit*, *del-insts*) ‹$g\text{-}iso\ g \wedge (\forall x.\ x \in norms \longrightarrow 0 \leq g\ x) \wedge bij\text{-}betw\ (\lambda x.\ if\ x \in norms\ then\ g\ x\ else\ undefined)\ norms\ \{r.\ 0 \leq r\}$› ‹$y \in norms \wedge z \in norms \wedge z < y$› *ax-space bij-betw-imp-inj-on calculation*(*3*) *calculation*(*4*) *g-iso-def in-mono inf-sup-ord*(*3*) *inj-on-def nonzero-eq-divide-eq norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms*(*1*) *vector-space-with-domain.smult-closed vector-space-with-domain.zero-in-dom*)
moreover have $otimes'\ ?beta\ (norm'\ x0) = z$
by (*smt* (*verit*, *del-insts*) ‹$g\text{-}iso\ g \wedge (\forall x.\ x \in norms \longrightarrow 0 \leq g\ x) \wedge bij\text{-}betw$

($\lambda x.$ *if* $x \in$ *norms then g x else undefined*) *norms* $\{r.\ 0 \leq r\}$› ‹*xx0* $=$ *norm′ x0*› ‹*xx0* $\in$ *norms* $\wedge$ *xx0* $\neq 0$› ‹*y* $\in$ *norms* $\wedge$ *z* $\in$ *norms* $\wedge$ $z < y$› *ax-space bij-betw-imp-inj-on g-iso-def inf-sup-ord(3) inj-on-def nonzero-eq-divide-eq norms-all-def norms-def norms-neg-def one-dim-vector-space-with-domain.axioms(1) subset-iff vector-space-with-domain.smult-closed vector-space-with-domain.zero-in-dom*)

moreover have *?alpha* $\geq 0 \wedge$ *?beta* ≥ 0

using ‹*g-iso g* $\wedge$ ($\forall x.\ x \in$ *norms* $\longrightarrow 0 \leq g\ x$) $\wedge$ *bij-betw* ($\lambda x.$ *if* $x \in$ *norms then g x else undefined*) *norms* $\{r.\ 0 \leq r\}$› ‹*xx0* $=$ *norm′ x0*› ‹*xx0* $\in$ *norms* $\wedge$ *xx0* $\neq 0$› ‹$y \in$ *norms* $\wedge$ $z \in$ *norms* $\wedge$ $z < y$› **by** *auto*

moreover have $0 <$ *?alpha* $\wedge$ *?alpha* $<$ *?beta* $\longleftrightarrow 0<y \wedge y<z$

by (*smt* (*verit*, *ccfv-threshold*) ‹$\forall \alpha\ \beta\ x.\ 0 \leq \alpha \wedge \alpha \leq \beta \longrightarrow$ *otimes′* α (*norm′ x*) $\leq$ *otimes′* β (*norm′ x*)› ‹$y \in$ *norms* $\wedge$ $z \in$ *norms* $\wedge$ $z < y$› *calculation(1) calculation(2)*)

moreover have $0<$*?alpha* $\wedge$ *?alpha* $<$ *?beta* $\longleftrightarrow 0 \leq$ (*?f y*) $\wedge$ (*?f y*) $<$ (*?f z*)

by (*smt* (*verit*, *best*) ‹$\forall \alpha\ \beta\ x.\ 0 \leq \alpha \wedge \alpha \leq \beta \longrightarrow$ *otimes′* α (*norm′ x*) $\leq$ *otimes′* β (*norm′ x*)› ‹$y \in$ *norms* $\wedge$ $z \in$ *norms* $\wedge$ $z < y$› *calculation(1) calculation(2) calculation(3) div-by-0 frac-less zero-le-divide-iff*)

ultimately show *?thesis*

using ‹*g-iso g* $\wedge$ ($\forall x.\ x \in$ *norms* $\longrightarrow 0 \leq g\ x$) $\wedge$ *bij-betw* ($\lambda x.$ *if* $x \in$ *norms then g x else undefined*) *norms* $\{r.\ 0 \leq r\}$› ‹$y \in$ *norms* $\wedge$ $z \in$ *norms* $\wedge$ $z < y$› **by** *auto*

qed

qed

qed

qed

moreover have ($\forall x\ y.$ *?f* (*norm′* ($x \oplus y$)) $\leq$ *?f* (*norm′ x*) $+$ *?f* (*norm′ y*))

proof

fix *x*

show $\forall y.$ (*?f* (*norm′* ($x \oplus y$)) $\leq$ *?f* (*norm′ x*) $+$ *?f* (*norm′ y*))

proof

fix *y*

show (*?f* (*norm′* ($x \oplus y$)) $\leq$ *?f* (*norm′ x*) $+$ *?f* (*norm′ y*))

proof−

have *norm′ x*$\in$*norms*

using *norms-def* **by** *blast*

moreover have *norm′ y* $\in$ *norms*

using *norms-def* **by** *blast*

moreover have *norm′* ($x \oplus y$)$\in$ *norms*

using *norms-def* **by** *blast*

moreover have *norm′* ($x \oplus y$) $\leq$ *oplus′* (*norm′ x*) (*norm′ y*)

using *ax3* **by** *blast*

moreover have (*?f* (*norm′* ($x \oplus y$))) $\leq$ (*?f* (*oplus′* (*norm′ x*) (*norm′ y*)))

proof−

have *norm′* ($x \oplus y$) $\leq$ *oplus′* (*norm′ x*) (*norm′ y*) $\vee$ *norm′* ($x \oplus y$) $=$ *oplus′* (*norm′ x*) (*norm′ y*)

using *calculation(4)* **by** *blast*

moreover {

assume *st1*:*norm′* ($x \oplus y$) $<$ *oplus′* (*norm′ x*) (*norm′ y*)

have *norm′ x* ∈ *norms*
using *norms-def* **by** *blast*
moreover have *norm′ y* ∈ *norms*
using *norms-def* **by** *blast*
moreover have *vector-space-with-domain norms-all oplus′ 0 otimes′*
using *ax-space norms-def*
one-dim-vector-space-with-domain-def
by (*metis norms-all-def norms-neg-def*)
moreover have *oplus′* (*norm′ x*) (*norm′ y*) ∈ *norms-all*
by (*metis Un-iff calculation*(*1*) *calculation*(*2*) *calculation*(*3*) *norms-all-def vector-space-with-domain.add-closed*)
moreover have *st2*:*norm′* (*x* ⊕ *y*)∈ *norms*
by (*simp add*: ‹*norm′* (*x* ⊕ *y*) ∈ *norms*›)
moreover have *st3*:*oplus′* (*norm′ x*) (*norm′ y*) ∈ *norms*
using *ax3 calculation*(*4*) *comparing-norms-help st2* **by** *blast*

moreover have (*?f* (*norm′* (*x* ⊕ *y*))) < (*?f* (*oplus′* (*norm′ x*) (*norm′ y*)))
using *mon st1 st2 st3*
by *blast*
ultimately have *?thesis*
by *linarith*
}
moreover {
assume *norm′* (*x* ⊕ *y*) = *oplus′* (*norm′ x*) (*norm′ y*)
then have *?thesis*
by *auto*
}
ultimately show *?thesis*
by *fastforce*
qed
moreover have (*?f* (*oplus′* (*norm′ x*) (*norm′ y*))) = (*?f* (*norm′ x*)) + (*?f* (*norm′ y*))
proof−
have *f1*:*norm′* (*x* ⊕ *y*) ≤ *oplus′* (*norm′ x*) (*norm′ y*)
using *ax3* **by** *blast*
moreover have *f2*:*norm′* (*x* ⊕ *y*) ∈ *norms*
by (*simp add*: ‹*norm′* (*x* ⊕ *y*) ∈ *norms*›)
moreover have *f3*:*vector-space-with-domain norms-all oplus′ 0 otimes′*
using *ax-space norms-def*
one-dim-vector-space-with-domain-def
by (*metis norms-all-def norms-neg-def*)
moreover have *oplus′* (*norm′ x*) (*norm′ y*)∈ *norms*
by (*metis UnI1* ‹*norm′ x* ∈ *norms*› ‹*norm′ y* ∈ *norms*› *f1 f2 f3 normed-gyrolinear-space′′.comparing-norms-help normed-gyrolinear-space′′-axioms norms-all-def vector-space-with-domain.add-closed*)
ultimately show *?thesis*
using ‹*g-iso g* ∧ (∀ *x*. *x* ∈ *norms* ⟶ *0* ≤ *g x*) ∧ *bij-betw* (λ*x*. *if x* ∈ *norms then g x else undefined*) *norms* {*r*. *0* ≤ *r*}› ‹*norm′ x* ∈ *norms*› ‹*norm′ y* ∈ *norms*› *g-iso-def norms-all-def* **by** *force*

        qed
        ultimately show *?thesis*
          by *force*
      qed
    qed
  qed

  moreover have $(\forall\, r{::}real.\ (\forall\, x.\ (?f\ (norm'\ (r \otimes x)) = |r| * (?f\ (norm'\ x)))))$
    by (*smt* (*verit*, *ccfv-SIG*) *Un-iff* ‹*g-iso g* ∧ (∀ *x*. *x* ∈ *norms* ⟶ *0* ≤ *g x*) ∧ *bij-betw* (λ*x*. *if x* ∈ *norms then g x else undefined*) *norms* {*r*. *0* ≤ *r*}› *g-iso-def normed-gyrolinear-space''-axioms normed-gyrolinear-space''-def norms-all-def norms-def rangeI*)

  ultimately show *?thesis*
    using ‹*g-iso g* ∧ (∀ *x*. *x* ∈ *norms* ⟶ *0* ≤ *g x*) ∧ *bij-betw* (λ*x*. *if x* ∈ *norms then g x else undefined*) *norms* {*r*. *0* ≤ *r*}› **by** *blast*
qed

end

end
theory *GyroVectorSpaceIsomorphism*
  imports *GyroVectorSpace*
begin

locale *gyrocarrier-isomorphism'* =
        gyrocarrier-norms-embed' to-carrier +
        gyrocarrier to-carrier +
   G: *gyrocarrier-norms-embed' to-carrier'*
  for *to-carrier* :: *'a*::*gyrocommutative-gyrogroup* ⇒ *'b*::{*real-inner*, *real-normed-algebra-1*}
and
     to-carrier' :: *'c*::*gyrocommutative-gyrogroup* ⇒ *'d*::{*real-inner*, *real-normed-algebra-1*}
+
 fixes φ :: *'a* ⇒ *'c*
begin

definition φ_R :: *real* ⇒ *real* **where**
  $\varphi_R\ x = G.to\text{-}real'\ (\varphi\ (of\text{-}real'\ x))$

end

locale *gyrocarrier-isomorphism* = *gyrocarrier-isomorphism'* +
  assumes φ*bij* [*simp*]:
    bij φ
  assumes φ*plus* [*simp*]:

$\bigwedge u\ v :: 'a.\ \varphi\ (u \oplus v) = \varphi\ u \oplus \varphi\ v$
assumes *φinner-unit*:
$\bigwedge u\ v :: 'a.\ [\![u \neq 0_g;\ v \neq 0_g]\!] \Longrightarrow$
inner (*to-carrier′* (φ *u*) $/_R$ *G.gyronorm* (φ *u*)) (*to-carrier′* (φ *v*) $/_R$ *G.gyronorm* (φ *v*)) =
inner (*to-carrier u* $/_R$ *gyronorm u*) (*to-carrier v* $/_R$ *gyronorm v*)
assumes *φRgyronorm* [*simp*]:
$\bigwedge a.\ \varphi_R\ (gyronorm\ a) = G.gyronorm\ (\varphi\ a)$
begin

lemma *φinvφ* [*simp*]:
shows φ (*inv* φ *a*) = *a*
by (*meson φbij bij-inv-eq-iff*)

lemma *invφφ* [*simp*]:
shows (*inv* φ) (φ *a*) = *a*
by (*metis φbij bij-inv-eq-iff*)

lemma *φzero* [*simp*]:
shows $\varphi\ 0_g = 0_g$
by (*metis φplus gyro-left-cancel gyro-right-id*)

lemma *φminus* [*simp*]:
shows φ (⊖ *a*) = ⊖ (φ *a*)
by (*metis φplus φzero gyro-equation-left gyro-left-inv*)

lemma *invφplus*[*simp*]:
shows (*inv* φ)(*a* ⊕ *b*) = *inv* φ *a* ⊕ *inv* φ *b*
by (*metis φbij φplus bij-inv-eq-iff*)

lemma *φgyr* [*simp*]:
shows φ (*gyr u v a*) = *gyr* (φ *u*) (φ *v*) (φ *a*)
by (*simp add*: *gyr-def*)

lemma *invφgyr* [*simp*]:
shows (*inv* φ) (*gyr u v a*) = *gyr* (*inv* φ *u*) (*inv* φ *v*) (*inv* φ *a*)
by (*metis φbij φgyr bij-inv-eq-iff*)

lemma *φinner*:
assumes $u \neq 0_g\ v \neq 0_g$
shows *G.gyroinner* (φ *u*) (φ *v*) =
(*G.gyronorm* (φ *u*) / *gyronorm u*) $*_R$ (*G.gyronorm* (φ *v*) / *gyronorm v*) $*_R$ *gyroinner u v*
proof−
have $\varphi\ u \neq 0_g\ \varphi\ v \neq 0_g$
by (*metis φbij φzero assms*(*1*) *bij-inv-eq-iff*)
(*metis φbij φzero assms*(*2*) *bij-inv-eq-iff*)
then have *G.gyronorm* (φ *u*) ≠ *0* *G.gyronorm* (φ *v*) ≠ *0*
using *G.norm-zero-iff*

by *blast+*
then have *inner* (*to-carrier′* (*φ u*)) (*to-carrier′* (*φ v*)) =
G.gyronorm (*φ u*) $*_R$ *G.gyronorm* (*φ v*) $*_R$ *inner* (*to-carrier′* (*φ u*) $/_R$ *G.gyronorm* (*φ u*)) (*to-carrier′* (*φ v*) $/_R$ *G.gyronorm* (*φ v*))
by (*smt* (*verit*, *ccfv-threshold*) *divideR-right inner-commute inner-scaleR-right real-scaleR-def*)
also have ... = *G.gyronorm* (*φ u*) $*_R$ *G.gyronorm* (*φ v*) $*_R$ *inner* (*to-carrier u* $/_R$ *gyronorm u*) (*to-carrier v* $/_R$ *gyronorm v*)
using *φinner-unit*[*OF assms*]
by *simp*
finally show *?thesis*
unfolding *G.gyroinner-def gyroinner-def*
by (*simp add*: *divide-inverse-commute*)
qed

lemma *gyronorm′gyr*:
shows *G.gyronorm* (*gyr u v a*) = *G.gyronorm a*
proof (*cases a* = 0_g)
case *True*
then show *?thesis*
by (*simp add*: *gyr-id-3*)
next
case *False*
then show *?thesis*
by (*metis* $φ_R$*gyronorm φinvφ invφgyr norm-gyr*)
qed

end

sublocale *gyrocarrier-isomorphism* ⊆ *gyrocarrier to-carrier′*
proof
fix *u v a b*
show *G.gyroinner* (*gyr u v a*) (*gyr u v b*) = *G.gyroinner a b*
proof (*cases a* = 0_g ∨ *b* = 0_g)
case *True*
then show *?thesis*
by (*metis G.gyroinner-def G.to-carrier-zero gyr-id-3 inner-zero-left inner-zero-right*)
next
case *False*

let *?Ga* = *gyr* (*inv φ u*) (*inv φ v*) (*inv φ a*) **and** *?Gb* = *gyr* (*inv φ u*) (*inv φ v*) (*inv φ b*)

have *G.gyroinner* (*gyr u v a*) (*gyr u v b*) = *G.gyroinner* (*φ ?Ga*) (*φ ?Gb*)
using *invφgyr*
by *simp*
also have ... = (*G.gyronorm* (*φ ?Ga*) / ⟪*?Ga*⟫) $*_R$ (*G.gyronorm* (*φ ?Gb*) / ⟪*?Gb*⟫) $*_R$ *?Ga* · *?Gb*
proof (*subst φinner*)

show *gyr* (*inv* φ *u*) (*inv* φ *v*) (*inv* φ *a*) $\neq$ 0_g
by (*metis False* φ*inv*φ φ*zero gyr-id-3 gyr-inj*)
next
show *gyr* (*inv* φ *u*) (*inv* φ *v*) (*inv* φ *b*) $\neq$ 0_g
by (*metis False* φ*inv*φ φ*zero gyr-id-3 gyr-inj*)
qed *simp*
also have ... = (*G.gyronorm* (φ *?Ga*) / ⟪*?Ga*⟫) $*_R$ (*G.gyronorm* (φ *?Gb*) / ⟪*?Gb*⟫) $*_R$ (*inv* φ *a*) $\cdot$ (*inv* φ *b*)
using *inner-gyroauto-invariant*
by *presburger*
finally have *G.gyroinner* (*gyr u v a*) (*gyr u v b*) = (*G.gyronorm* (*gyr u v a*) / *gyronorm ?Ga*) $*_R$ (*G.gyronorm* (*gyr u v b*) / *gyronorm ?Gb*) $*_R$ (*inv* φ *a*) $\cdot$ (*inv* φ *b*)
by *auto*

moreover

have *G.gyroinner a b* = (*G.gyronorm a* / ⟪*inv* φ *a*⟫) $*_R$ (*G.gyronorm b* / ⟪*inv* φ *b*⟫) $*_R$ *inv* φ *a* $\cdot$ *inv* φ *b*
using φ*inner*[*of inv* φ *a inv* φ *b*]
by (*metis False* φ*inv*φ φ*zero*)

moreover
have ⟪*gyr* (*inv* φ *u*) (*inv* φ *v*) (*inv* φ *a*)⟫ = ⟪*inv* φ *a*⟫
⟪*gyr* (*inv* φ *u*) (*inv* φ *v*) (*inv* φ *b*)⟫ = ⟪*inv* φ *b*⟫
by (*auto simp add*: *norm-gyr*)

moreover

have *G.gyronorm* (*gyr u v a*) = *G.gyronorm a*
G.gyronorm (*gyr u v b*) = *G.gyronorm b*
using *gyronorm'gyr*
by *auto*

ultimately

show *?thesis*
by *simp*
qed
qed

locale *pre-gyrovector-space-isomorphism'* =
pre-gyrovector-space to-carrier scale +
gyrocarrier-norms-embed' to-carrier +
GC: *gyrocarrier-norms-embed' to-carrier'*
for *to-carrier* :: *'a*::*gyrocommutative-gyrogroup* ⇒ *'b*::{*real-inner*, *real-normed-algebra-1*}
and
to-carrier' :: *'c*::*gyrocommutative-gyrogroup* ⇒ *'d*::{*real-inner*, *real-normed-algebra-1*}
and

```
    scale :: real ⇒ 'a ⇒ 'a and
    scale' :: real ⇒ 'c ⇒ 'c +
  fixes φ :: 'a ⇒ 'c

sublocale pre-gyrovector-space-isomorphism' ⊆ gyrocarrier-isomorphism'
  ..

locale pre-gyrovector-space-isomorphism =
   pre-gyrovector-space-isomorphism' +
   gyrocarrier-isomorphism +
   assumes φscale [simp]:
      ⋀ r :: real. ⋀ u :: 'a. φ (scale r u) = scale' r (φ u)
begin

lemma scale'-1:
  shows scale' 1 a = a
  using φscale[of 1 (inv φ) a]
  by (simp add: scale-1)

lemma scale'-distrib:
  shows scale' (r1 + r2) a = scale' r1 a ⊕ scale' r2 a
  using φscale[symmetric, of r1 + r2 (inv φ) a]
  using scale-distrib
  by auto

lemma scale'-assoc:
  shows scale' (r1 * r2) a = scale' r1 (scale' r2 a)
  using φscale[symmetric, of r1 * r2 (inv φ) a]
  using scale-assoc
  by force

lemma scale'-gyroauto-id:
  shows gyr (scale' r1 v) (scale' r2 v) = id
proof
  fix x
  show gyr (scale' r1 v) (scale' r2 v) x = id x
    using φscale[symmetric, of r1 (inv φ) v] φscale[symmetric, of r2 (inv φ) v]
    by (metis φinvφ gyroauto-id id-apply invφφ invφgyr)
qed

lemma scale'-gyroauto-property:
  shows gyr u v (scale' r a) = scale' r (gyr u v a)
  using φscale[of r inv φ (gyr u v a)]
  using φscale[of r inv φ a]
  by (metis φbij bij-inv-eq-iff gyroauto-property invφgyr)

end

locale gyrovector-space-isomorphism' =
```

pre-gyrovector-space-isomorphism +
gyrovector-space-norms-embed scale +
GC: *gyrocarrier-norms-embed to-carrier′ scale′* +
assumes *φreals*:
φ ‘ *reals* = *GC.reals*
begin

lemma φ_R*norms*:
assumes $a \in$ *norms*
shows φ_R $a \in$ *GC.norms*
using *assms*
by (*metis GC.bij-betw-to-real′* φ_R*-def φreals bij-betw-iff-bijections bij-reals-norms image-iff*)

lemma *φof-real′*[*simp*]:
assumes $a \in$ *norms*
shows φ (*of-real′ a*) = *GC.of-real′* (φ_R *a*)
using *assms*
by (*metis GC.of-real′* φ_R*-def φreals bij-betwE bij-reals-norms image-iff*)

lemma *φgyronorm* [*simp*]:
shows φ (*of-real′* (*gyronorm a*)) = *GC.of-real′* (*GC.gyronorm* (φ *a*))
by *simp*

lemma φ_R*plus* [*simp*]:
assumes $a \in$ *norms* $b \in$ *norms*
shows φ_R $(a \oplus_R b)$ = *GC.oplusR* (φ_R *a*) (φ_R *b*)
proof−
have φ_R $(a \oplus_R b)$ = *GC.to-real′* (φ (*of-real′ a*) ⊕ φ (*of-real′ b*))
unfolding φ_R*-def oplusR-def*
proof (*subst of-real′*)
show *of-real′ a* ⊕ *of-real′ b* ∈ *reals*
by (*meson assms*(*1*) *assms*(*2*) *bij-betwE bij-reals-norms oplus-reals*)
qed *simp*
then show *?thesis*
using *assms*
unfolding *GC.oplusR-def*
by *simp*
qed

lemma φ_R*plus′* [*simp*]:
φ_R ((⟪*a*⟫) $\oplus_R$ (⟪*b*⟫)) = *GC.oplusR* (φ_R (⟪*a*⟫)) (φ_R (⟪*b*⟫))
by (*simp add*: *GC.oplusR-def* φ_R*-def*)

lemma φ_R*times* [*simp*]:
assumes $a \in$ *norms*
shows φ_R $(r \otimes_R a)$ = *GC.otimesR r* (φ_R *a*)
proof−
have φ_R $(r \otimes_R a)$ = *GC.to-real′* (φ (*of-real′* (*to-real′* (*scale r* (*of-real′ a*)))))

unfolding φ_R-*def otimesR-def*
by *simp*
also have $\ldots = GC.to\text{-}real'\ (\varphi\ (scale\ r\ (of\text{-}real'\ a)))$
using *assms*
by (*metis bij-betwE bij-reals-norms of-real′ otimes-reals*)
finally show *?thesis*
using *assms*
unfolding *GC.otimesR-def*
by *simp*
qed

lemma $\varphi_R times'$ [*simp*]:
shows $\varphi_R\ (r \otimes_R (\langle\!\langle a\rangle\!\rangle)) = GC.otimesR\ r\ (\varphi_R\ (\langle\!\langle a\rangle\!\rangle))$
by (*simp add*: *GC.otimesR-def* φ_R-*def*)

lemma $\varphi_R inv$ [*simp*]:
assumes $a \in norms$
shows $\varphi_R\ (\ominus_R\ a) = GC.oinvR\ (\varphi_R\ a)$
proof−
have $\varphi_R\ (\ominus_R\ a) = GC.to\text{-}real'\ (\varphi\ (of\text{-}real'\ (to\text{-}real'\ (\ominus\ (of\text{-}real'\ a)))))$
unfolding φ_R-*def oinvR-def*
by *simp*
also have $\ldots = GC.to\text{-}real'\ (\varphi\ (\ominus\ (of\text{-}real'\ a)))$
by (*metis assms bij-betwE bij-reals-norms of-real′ oinv-reals*)
finally show *?thesis*
using *assms*
unfolding *GC.oinvR-def*
by *simp*
qed

lemma $\varphi_R inv'$ [*simp*]:
$\varphi_R\ (\ominus_R\ (\langle\!\langle a\rangle\!\rangle)) = GC.oinvR\ (\varphi_R\ (\langle\!\langle a\rangle\!\rangle))$
by (*simp add*: φ_R-*def GC.oinvR-def*)

lemma *scale′-prop1′*:
assumes $u \neq 0_g$ $r \neq 0$
shows $to\text{-}carrier'\ (\varphi\ (scale\ |r|\ u))\ /_R\ GC.gyronorm\ (\varphi\ (scale\ |r|\ u)) =$
$(to\text{-}carrier'\ (\varphi\ u)\ /_R\ GC.gyronorm\ (\varphi\ u))$ (**is** $?a = ?b$)
proof−
have $\langle\!\langle scale\ r\ u\rangle\!\rangle = \langle\!\langle scale\ (abs\ r)\ u\rangle\!\rangle$
using *homogeneity* **by** *force*

have $inner\ ?b\ ?a =$
$inner\ (to\text{-}carrier\ u\ /_R\ \langle\!\langle u\rangle\!\rangle)\ (to\text{-}carrier\ (scale\ |r|\ u)\ /_R\ \langle\!\langle scale\ |r|\ u\rangle\!\rangle)$
using φ*inner-unit*[**where** $u{=}u$ **and** $v{=}scale\ (abs\ r)\ u$]
by *fastforce*
also have $\ldots = inner\ (to\text{-}carrier\ u\ /_R\ \langle\!\langle u\rangle\!\rangle)\ (to\text{-}carrier\ u\ /_R\ \langle\!\langle u\rangle\!\rangle)$
using *scale-prop1*[*of r u*] $\langle u \neq 0_g\rangle$ $\langle r \neq 0\rangle$ $\langle\langle\!\langle scale\ r\ u\rangle\!\rangle = \langle\!\langle scale\ (abs\ r)\ u\rangle\!\rangle\rangle$

by *simp*
finally have *inner ?b ?a* = *1*
by (*smt* (*verit*, *ccfv-threshold*) φ*inner-unit assms*(*1*) *gyronorm-def inverse-nonnegative-iff-nonnegative inverse-nonzero-iff-nonzero left-inverse norm-eq norm-eq-1 norm-le-zero-iff norm-scaleR norm-zero-iff scaleR-eq-0-iff to-carrier-zero-iff*)

show *?thesis*
proof (*rule inner-eq-1*)
show *inner ?a ?b* = *1*
using ‹*inner ?b ?a* = *1*›
by (*simp add*: *inner-commute*)
next
show *norm ?a* = *1*
by (*smt* (*verit*, *ccfv-threshold*) φ*inner-unit assms*(*1*) *assms*(*2*) *gyronorm-def scale-prop1 inverse-nonnegative-iff-nonnegative inverse-nonzero-iff-nonzero left-inverse local.norm-zero norm-eq norm-ge-zero norm-scaleR norm-zero-iff scaleR-eq-0-iff to-carrier-zero-iff*)
next
show *norm ?b* = *1*
using *GC.norm-zero-iff* φ*zero assms*(*1*) *GC.gyronorm-def inv*$\varphi\varphi$ *inverse-negative-iff-negative left-inverse norm-not-less-zero norm-scaleR*
by (*smt* (*verit*, *del-insts*))
qed
qed

lemma *scale'-prop1*:
assumes $a \neq 0_g$ $r \neq 0$
shows *to-carrier'* (*scale'* $|r|$ *a*) $/_R$ *GC.gyronorm* (*scale' r a*) = *to-carrier' a* $/_R$ *GC.gyronorm a*
proof−
from *assms* **have** ∗: *inv* φ $a \neq 0_g$ $r \neq 0$
by (*metis* φ*inv*φ φ*zero*, *simp*)
show *?thesis*
using *scale'-prop1'*[*OF* ∗]
by (*metis* φ_R*gyronorm* φ*inv*φ φ*scale abs-idempotent homogeneity*)
qed

lemma *scale'-homogeneity*:
shows *GC.gyronorm* (*scale' r a*) = *GC.otimesR* $|r|$ (*GC.gyronorm a*)
proof−
have *GC.gyronorm* (*scale' r a*) = *GC.gyronorm* (φ (*scale r* (*inv* φ *a*)))
using φ*scale*[*of r inv* φ *a*]
by *simp*
also have ... = φ_R (*gyronorm* (*scale r* (*inv* φ *a*)))
by *simp*
also have ... = φ_R ($|r| \otimes_R$ (《*inv* φ *a*》))
using *homogeneity*
by *simp*
also have ... = φ_R (*to-real'* (*scale* $|r|$ (*of-real'* (《*inv* φ *a*》))))
unfolding *otimesR-def*

by *simp*
also have ... $=$ *GC.to-real′* (φ (*scale* $|r|$ (*of-real′* (*gyronorm* (*inv* φ *a*)))))
using *GC.otimesR-def* φ_R*times otimesR-def*
by *force*
also have ... $=$ *GC.to-real′* (*scale′* $|r|$ (φ (*of-real′* (*gyronorm* (*inv* φ *a*)))))
using φ*scale*
by *simp*
also have ... $=$ *GC.to-real′* (*scale′* $|r|$ (*GC.of-real′* (*GC.gyronorm* (φ (*inv* φ *a*)))))
by (*subst* φ*gyronorm*, *simp*)
finally
show *GC.gyronorm* (*scale′* *r* *a*) $=$ *GC.otimesR* $|r|$ (*GC.gyronorm* *a*)
unfolding *GC.otimesR-def*
by *simp*
qed

end

sublocale *gyrovector-space-isomorphism′* $\subseteq$ *GV*: *pre-gyrovector-space to-carrier′ scale′*
by (*meson gyrocarrier-axioms scale′-prop1 pre-gyrovector-space.intro pre-gyrovector-space-axioms.intro scale′-1 scale′-assoc scale′-distrib scale′-gyroauto-id scale′-gyroauto-property*)

locale *gyrovector-space-isomorphism* $=$
gyrovector-space-isomorphism′ $+$
assumes φ_R*mono*:
$\bigwedge$ *a b*. $[\![a \in norms;\ b \in norms;\ 0 \le a;\ a \le b]\!] \Longrightarrow \varphi_R\ a \le \varphi_R\ b$
begin

lemma *scale′-triangle*:
shows *GC.gyronorm* ($a \oplus b$) $\le$ *GC.oplusR* (*GC.gyronorm* *a*) (*GC.gyronorm* *b*)
proof−
have *GC.gyronorm* ($a \oplus b$) $= \varphi_R$ ($\langle\!\langle$*inv* φ *a* $\oplus$ *inv* φ *b*$\rangle\!\rangle$)
by *simp*
moreover
have *GC.oplusR* (*GC.gyronorm* *a*) (*GC.gyronorm* *b*) $= \varphi_R$ (($\langle\!\langle$*inv* φ *a*$\rangle\!\rangle$) $\oplus_R$ ($\langle\!\langle$*inv* φ *b*$\rangle\!\rangle$))
by *simp*
moreover
have φ_R ($\langle\!\langle$*inv* φ *a* $\oplus$ *inv* φ *b*$\rangle\!\rangle$) $\le \varphi_R$ (($\langle\!\langle$*inv* φ *a*$\rangle\!\rangle$) $\oplus_R$ ($\langle\!\langle$*inv* φ *b*$\rangle\!\rangle$))
proof (*rule* φ_R*mono*)
show $\langle\!\langle$*inv* φ *a* $\oplus$ *inv* φ *b*$\rangle\!\rangle \in$ *norms*
unfolding *norms-def*
by *auto*
next
show $\langle\!\langle$*inv* φ *a*$\rangle\!\rangle \oplus_R$ ($\langle\!\langle$*inv* φ *b*$\rangle\!\rangle$) $\in$ *norms*
proof (*rule oplusR-norms*)
show $\langle\!\langle$*inv* φ *a*$\rangle\!\rangle \in$ *norms* $\langle\!\langle$*inv* φ *b*$\rangle\!\rangle \in$ *norms*
unfolding *norms-def*

by *auto*
qed
next
show $0 \leq \langle\langle inv\ \varphi\ a \oplus inv\ \varphi\ b\rangle\rangle$
by (*simp add*: *gyronorm-def*)
next
show $\langle\langle inv\ \varphi\ a \oplus inv\ \varphi\ b\rangle\rangle \leq (\langle\langle inv\ \varphi\ a\rangle\rangle) \oplus_R (\langle\langle inv\ \varphi\ b\rangle\rangle)$
by (*simp add*: *gyrotriangle*)
qed
ultimately
show $GC.gyronorm\ (a \oplus b) \leq GC.oplusR\ (GC.gyronorm\ a)\ (GC.gyronorm\ b)$
by *simp*
qed

end

sublocale *gyrovector-space-isomorphism* ⊆ *gyrovector-space-norms-embed scale′ to-carrier′*
by (*meson GC.gyrocarrier-norms-embed-axioms GV.pre-gyrovector-space-axioms gyrocarrier-axioms gyrovector-space-isomorphism.scale′-triangle gyrovector-space-isomorphism-axioms gyrovector-space-norms-embed-axioms.intro gyrovector-space-norms-embed-def scale′-homogeneity*)

locale *gyrocarrier-isomorphism-norms-embed′* = *gyrovector-space-norms-embed scale to-carrier* +
GC: *gyrocarrier-norms-embed′ to-carrier′*
for *to-carrier* :: ′*a*::*gyrocommutative-gyrogroup* ⇒ ′*b*::{*real-inner*, *real-normed-algebra-1*}
and
to-carrier′ :: ′*c*::*gyrocommutative-gyrogroup* ⇒ ′*d*::{*real-inner*, *real-normed-algebra-1*}
and
scale :: *real* ⇒ ′*a* ⇒ ′*a* +
fixes *scale′* :: *real* ⇒ ′*c* ⇒ ′*c*
fixes φ :: ′*a* ⇒ ′*c*
begin

definition φ_R :: *real* ⇒ *real* **where**
$\varphi_R\ x = GC.to\text{-}real'\ (\varphi\ (of\text{-}real'\ x))$

end

locale *gyrocarrier-isomorphism-norms-embed* = *gyrocarrier-isomorphism-norms-embed′* +
assumes φ*bij*:
bij φ
assumes φ*plus* [*simp*]:
$\bigwedge u\ v :: 'a.\ \varphi\ (u \oplus v) = \varphi\ u \oplus \varphi\ v$
assumes φ*scale* [*simp*]:
$\bigwedge r :: real.\ \bigwedge u :: 'a.\ \varphi\ (scale\ r\ u) = scale'\ r\ (\varphi\ u)$
assumes φ*reals*:

φ ‘ *reals* $=$ *GC.reals*
assumes φ_R*gyronorm* [*simp*]:
$\bigwedge$ *a*. φ_R (*gyronorm a*) $=$ *GC.gyronorm* (φ *a*)
assumes *GCoinvRminus*:
$\bigwedge$ *a*. *a* $\in$ *GC.norms* $\Longrightarrow$ *GC.oinvR a* $= -a$
begin

lemma φ*inv*φ [*simp*]:
shows φ (*inv* φ *a*) $=$ *a*
by (*meson* φ*bij bij-inv-eq-iff*)

lemma φ*zero* [*simp*]:
shows φ $0_g = 0_g$
by (*metis* φ*plus gyro-left-cancel gyro-right-id*)

lemma φ*minus* [*simp*]:
shows φ ($\ominus$ *a*) $= \ominus$ (φ *a*)
by (*metis* φ*plus* φ*zero gyro-equation-left gyro-left-inv*)

lemma φ*gyronorm* [*simp*]:
shows φ (*of-real*$'$ (*gyronorm a*)) $=$ *GC.of-real*$'$ (*GC.gyronorm* (φ *a*))
by (*metis* (*no-types*, *opaque-lifting*) *GC.of-real*$'$ φ_R-*def* φ_R*gyronorm* φ*reals bij-betwE bij-reals-norms image-eqI norm-in-norms*)

lemma φ_R*inv*$'$ [*simp*]:
φ_R ($\ominus_R$ ($\langle\!\langle a\rangle\!\rangle$)) $=$ *GC.oinvR* (φ_R ($\langle\!\langle a\rangle\!\rangle$))
by (*simp add*: *GC.oinvR-def* φ_R-*def*)
end

sublocale *gyrocarrier-isomorphism-norms-embed* $\subseteq$ *GV*$'$: *gyrocarrier-norms-embed to-carrier*$'$ *scale*$'$
proof
fix *a b*
assume *a* $\in$ *GC.reals b* $\in$ *GC.reals*
then obtain *x y* **where**
a $=$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\vee$ *a* $=$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$))
b $=$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$)) $\vee$ *b* $=$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$))
unfolding *GC.reals-def GC.norms-def GC.of-real*$'$-*def*
by *fastforce*
then have *a* $\oplus$ *b* $=$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\oplus$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$)) $\vee$
a $\oplus$ *b* $=$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\oplus$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$)) $\vee$
a $\oplus$ *b* $=$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\oplus$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$)) $\vee$
a $\oplus$ *b* $=$ *GC.of-real*$'$ ($-$ φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\oplus$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$))
by *auto*
then have *a* $\oplus$ *b* $=$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *x*$\rangle\!\rangle$)) $\oplus$ *GC.of-real*$'$ (φ_R ($\langle\!\langle$*inv* φ *y*$\rangle\!\rangle$)) $\vee$

$a \oplus b = GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle)) \oplus GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ (\langle\langle inv\ \varphi\ y\rangle\rangle))) \lor$
$a \oplus b = GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \oplus GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ ((\langle\langle inv\ \varphi\ y\rangle\rangle)))) \lor$
$a \oplus b = GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \oplus GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ y\rangle\rangle))$
by (*simp add*: *GCoinvRminus*)
then show $a \oplus b \in GC.reals$
by (*smt* (*verit*, *del-insts*) ‹$a \in GC.reals$› ‹$b \in GC.reals$› *φplus φreals image-iff oplus-reals*)
next
fix *a*
assume $a \in GC.reals$
then obtain *x* **where**
$a = GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle)) \lor a = GC.of\text{-}real'\ (-\ \varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))$
unfolding *GC.reals-def GC.norms-def GC.of-real′-def*
by *fastforce*
then have $\ominus\ a = \ominus\ (GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \lor$
$\ominus\ a = \ominus\ (GC.of\text{-}real'\ (-\ \varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle)))$
by *auto*
then have $\ominus\ a = \ominus\ (GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \lor$
$\ominus\ a = \ominus\ (GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))))$
by (*simp add*: *GCoinvRminus*)
then show $\ominus\ a \in GC.reals$
by (*metis* (*no-types*, *opaque-lifting*) ‹$a \in GC.reals$› *φminus φreals image-iff oinv-reals*)
next
fix *a r*
assume $a \in GC.reals$
then obtain *x* **where**
$a = GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle)) \lor a = GC.of\text{-}real'\ (-\ \varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))$
unfolding *GC.reals-def GC.norms-def GC.of-real′-def*
by *fastforce*
then have $scale'\ r\ a = scale'\ r\ (GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \lor$
$scale'\ r\ a = scale'\ r\ (GC.of\text{-}real'\ (-\ \varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle)))$
by *auto*
then have $scale'\ r\ a = scale'\ r\ (GC.of\text{-}real'\ (\varphi_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))) \lor$
$scale'\ r\ a = scale'\ r\ (GC.of\text{-}real'\ (\varphi_R\ (\ominus_R\ (\langle\langle inv\ \varphi\ x\rangle\rangle))))$
by (*simp add*: *GCoinvRminus*)
then show $scale'\ r\ a \in GC.reals$
by (*smt* (*verit*, *best*) ‹$a \in GC.reals$› *φreals φscale image-iff otimes-reals*)
qed

end
theory *MoreComplex*
imports *Complex-Main HOL−Analysis.Inner-Product*
begin

lemma *real-compex-cmod*:

fixes *r*::*real*
shows *cmod*(*r* ∗ *z*) = *abs r* ∗ *cmod z*
by (*metis abs-mult norm-of-real*)

lemma *cnj-closed-for-unit-disc*:
assumes *cmod z1* < *1*
shows *cmod* (*cnj z1*) <*1*
by (*simp add*: *assms*)

lemma *mult-closed-for-unit-disc*:
assumes *cmod z1* < *1 cmod z2* < *1*
shows *cmod* (*z1*∗*z2*) < *1*
using *assms*(*1*) *assms*(*2*) *norm-mult-less*
by *fastforce*

lemma *cnj-cmod*:
shows *z1* ∗ *cnj z1* = (*cmod z1*)^*2*
using *complex-norm-square*
by *fastforce*

lemma *cnj-cmod-1*:
assumes *cmod z1* = *1*
shows *z1* ∗ *cnj z1* = *1*
by (*metis assms complex-cnj-one complex-norm-square mult.right-neutral norm-one*)

lemma *den-not-zero*:
assumes *cmod a* < *1 cmod b* < *1*
shows *1* + *cnj a* ∗ *b* ≠ *0*
using *assms*
by (*smt add.inverse-unique complex-mod-cnj i-squared norm-ii norm-mult norm-mult-less*)

lemma *cmod-mix-cnj*:
assumes *cmod u* < *1 cmod v* < *1*
shows *cmod* ((*1* + *u*∗*cnj v*) / (*1* + *v*∗*cnj u*)) = *1*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*1*) *assms*(*2*) *complex-cnj-add complex-cnj-cnj complex-cnj-mult complex-cnj-one complex-mod-cnj den-not-zero divide-self-if mult.commute norm-divide norm-one*)

lemma *cnj-mix-ex-real-k*:
assumes *v* ≠ *0*
shows *x* ∗ *cnj v* = *v* ∗ *cnj x* ⟷ (∃ (*k*::*real*). *x* = *k* ∗ *v*)
proof−
have *vx*: *v* = *Re v* + *Im v* ∗ i *x* = *Re x* + *Im x* ∗ i
by (*simp add*: *complex-eq mult.commute*)+

have *x* ∗ *cnj v* = *v* ∗ *cnj x* ⟷ (*Re x* + *Im x* ∗ i) ∗ (*Re v* − *Im v* ∗ i) = (*Re v* + *Im v* ∗ i) ∗ (*Re x* − *Im x* ∗ i)
by (*metis complex-cnj-add complex-cnj-complex-of-real complex-cnj-i complex-cnj-mult complex-eq complex-of-real-i diff-conv-add-uminus i-complex-of-real mult-minus-left*)

also have ... ⟷ $(Re\ v * Im\ x - Re\ x * Im\ v) * \mathrm{i} =$
$(- Re\ v * Im\ x + Re\ x * Im\ v) * \mathrm{i}$
by (*simp add*: *field-simps*)
also have ... ⟷ $Re\ v * Im\ x = Re\ x * Im\ v$
by (*smt* (*verit*, *best*) *complex-i-not-zero mult-minus-left mult-right-cancel of-real-eq-iff*)
also have ... ⟷ $(\exists\ (k{::}real).\ x = k * v)$
proof (*cases Im v = 0*)
case *True*
then show *?thesis*
using *assms vx*
by (*smt* (*verit*, *best*) *Im-divide-of-real add.right-neutral calculation complex-cnj-complex-of-real complex-cnj-mult complex-eq mult.commute mult-eq-0-iff nonzero-mult-div-cancel-left of-real-0 times-divide-eq-right*)
next
case *False*
then have $Re\ v * Im\ x = Re\ x * Im\ v \longleftrightarrow x = (Im\ x\ /\ Im\ v) * v$
using *assms vx*
by (*smt* (*verit*, *ccfv-SIG*) *calculation complex-cnj-complex-of-real complex-cnj-mult complex-of-real-mult-Complex complex-surj mult.commute nonzero-mult-div-cancel-right times-divide-eq-left*)
then show *?thesis*
using *assms vx*
by (*smt* (*verit*, *del-insts*) *calculation complex-cnj-complex-of-real complex-cnj-mult mult.assoc mult.commute*)
qed
finally show *?thesis*
.
qed

lemma *two-inner-cnj*:
shows $2 * inner\ u\ v = cnj\ u * v + cnj\ v * u$
by (*smt* (*verit*) *cnj.simps*(*1*) *cnj.simps*(*2*) *cnj-add-mult-eq-Re inner-complex-def mult.commute mult-minus-left times-complex.simps*(*1*))

abbreviation *cor* ≡ *complex-of-real*

lemma *abs-inner-lt-1*:
assumes $norm\ u < 1$ $norm\ v < 1$
shows $abs\ (inner\ u\ v) < 1$
using *Cauchy-Schwarz-ineq2*[*of u v*]
by (*smt* (*verit*) *assms*(*1*) *assms*(*2*) *mult-le-cancel-left2 norm-not-less-zero*)

lemma *inner-lt-1*:
assumes $norm\ u < 1$ $norm\ v < 1$
shows $inner\ u\ v < 1$
using *assms*
using *abs-inner-lt-1*
by *fastforce*

```
lemma inner-def1:
  shows inner z1 z2 = (z1 * cnj z2 + z2 * cnj z1) / 2
proof-
  obtain a b where ab: Re z1 = a ∧ Im z1 = b
    by blast
  obtain c d where cd: Re z2 = c ∧ Im z2 = d
    by blast
  have Re (z1 * cnj z2) = a*c + b*d Re (z2 * cnj z1) = a*c + b*d
       Im (z1 * cnj z2) = b*c − a*d Im (z2 * cnj z1) = −b*c + a*d
    using ab cd
    by simp+
  then have (z1 * cnj z2 + z2 * cnj z1) / 2 =  a*c + b*d
    using complex-eq-iff by force
  then show ?thesis
    using ab cd inner-complex-def
    by presburger
qed

lemma inner-def2:
  shows inner z1 z2 = Re (cnj z1 * z2)
  by (simp add: inner-complex-def)


end
theory GammaFactor
  imports Complex-Main MoreComplex
begin

definition gamma-factor :: 'a::real-inner ⇒ real (γ) where
  γ u = (if norm u < 1 then
            1 / sqrt (1 − (norm u)²)
         else
            0)

lemma gamma-factor-nonzero:
  assumes norm u < 1
  shows 1 / sqrt (1 − (norm u)²) ≠ 0
  using assms square-norm-one by force

lemma gamma-factor-increasing:
  fixes t1 t2 ::real
  assumes 0 ≤ t2 t2 < t1 t1 < 1
  shows γ t2 < γ t1
proof-
  have d: cmod t1 = abs t1 cmod t2 = abs t2
    using norm-of-real
    by blast+
```

```
  have |t2| * |t2| < |t1| * |t1|
    by (simp add: assms mult-strict-mono')
  then have 1 − |t1| * |t1| < 1 − |t2| * |t2|
    by auto
  then have sqrt (1 − |t1| * |t1|) < sqrt (1 − |t2| * |t2|)
    using real-sqrt-less-iff
    by blast
  then have 1 / sqrt (1 − |t2| * |t2|) < 1 / sqrt (1 − |t1| * |t1|)
    using assms
    by (smt (z3) frac-less2 mult-less-cancel-right2 real-sqrt-gt-zero)

  moreover

  have norm t1 < 1 norm t2 < 1
    using assms
    by force+
  then have γ t1 = 1 / sqrt(1 − (abs t1)^2) γ t2 = 1 / sqrt(1 − (abs t2)^2)
    using assms d
    unfolding gamma-factor-def
    by auto

  ultimately

  show ?thesis
    using d
    by (metis power2-eq-square)
qed

lemma gamma-factor-increase-reverse:
  fixes t1 t2 :: real
  assumes t1 ≥ 0 t1 < 1 t2 ≥ 0 t2 < 1
  assumes γ t1 > γ t2
  shows t1 > t2
  using assms
  by (smt (verit, best) gamma-factor-increasing)

lemma gamma-factor-u-normu:
  fixes u :: real
  assumes 0 ≤ u u ≤ 1
  shows γ u = γ (norm u)
  unfolding gamma-factor-def
  by auto

lemma gamma-factor-positive:
  assumes norm u < 1
  shows γ u > 0
  using assms
  unfolding gamma-factor-def
 by (smt (verit, del-insts) divide-pos-pos norm-ge-zero power2-eq-square power2-nonneg-ge-1-iff
```

real-sqrt-gt-0-iff)

lemma *norm-square-gamma-factor*:
assumes $norm\ u < 1$
shows $(norm\ u)\hat{}2 = 1 - 1 / (\gamma\ u)\hat{}2$
proof−
have $\gamma\ u = 1 / sqrt\ (1 - (norm\ u)\hat{}2)$
by (*metis assms gamma-factor-def*)
then have $(\gamma\ u)\hat{}2 = 1 / (1 - (norm\ u)\hat{}2)$
using *assms*
by (*metis abs-power2 gamma-factor-positive less-eq-real-def norm-ge-zero norm-power power2-eq-imp-eq real-norm-def real-sqrt-abs real-sqrt-divide real-sqrt-eq-1-iff real-sqrt-eq-iff*)
then show *?thesis*
by *auto*
qed

lemma *norm-square-gamma-factor'*:
assumes $norm\ u < 1$
shows $(norm\ u)\hat{}2 = ((\gamma\ u)\hat{}2 - 1) / (\gamma\ u)\hat{}2$
using *norm-square-gamma-factor*[*OF assms*]
by (*metis assms diff-divide-distrib div-self gamma-factor-positive norm-not-less-zero norm-zero power-not-zero*)

lemma *gamma-factor-square-norm*:
assumes $norm\ u < 1$
shows $(\gamma\ u)^2 = 1 / (1 - (norm\ u)^2)$
by (*smt* (*verit*) *assms gamma-factor-def gamma-factor-positive real-sqrt-divide real-sqrt-eq-iff real-sqrt-one real-sqrt-unique*)

lemma *gamma-expression-eq-one-1*:
assumes $norm\ u < 1$
shows $1 / \gamma\ u + (\gamma\ u * (norm\ u)\hat{}2) / (1 + \gamma\ u) = 1$
proof−
have $\gamma\ u \neq 0$ $1 + \gamma\ u \neq 0$
using *assms gamma-factor-positive*
by *fastforce+*

have $1 / \gamma\ u + \gamma\ u * (norm\ u)\hat{}2 / (1 + \gamma\ u) =$
$(1 + \gamma\ u + (\gamma\ u)\hat{}2 * (norm\ u)\hat{}2) / (\gamma\ u * (1 + \gamma\ u))$
using ‹$\gamma\ u \neq 0$› ‹$1 + \gamma\ u \neq 0$›
by (*metis* (*no-types*, *lifting*) *add-divide-distrib nonzero-divide-mult-cancel-right nonzero-mult-divide-mult-cancel-left numeral-One power-add-numeral2 power-one-right semiring-norm*(*2*))
also have ... $= (1 + \gamma\ u + (\gamma\ u)\hat{}2 * (1 - 1 / ((\gamma\ u)\hat{}2))) / (\gamma\ u * (1 + \gamma\ u))$
by (*simp add*: *assms norm-square-gamma-factor*)
also have ... $= (1 + \gamma\ u + (\gamma\ u)\hat{}2 - 1) / (\gamma\ u * (1 + \gamma\ u))$
by (*simp add*: *Rings.ring-distribs*(*4*))

```
  also have ... = (γ u * (1 + γ u)) / (γ u * (1 + γ u))
    by (simp add: power2-eq-square ring-class.ring-distribs(1))
  finally show ?thesis
    using ‹γ u ≠ 0› ‹1 + γ u ≠ 0›
    by (metis div-by-1 divide-divide-eq-right eq-divide-eq-1)
qed

lemma gamma-expression-eq-one-2:
  assumes norm u < 1
  shows ((γ u)^2 * (norm u)^2) / (1 + γ u)^2 + (2 * γ u) / (γ u * (1 + γ u))
= 1
proof−
  have *: γ u ≠ 0 1 + γ u ≠ 0
    using assms gamma-factor-positive
    by force+

  have ((γ u)^2 * (norm u)^2) / (1 + γ u)^2 + (2 * γ u) / (γ u * (1 + γ u))
=
        ((γ u)^2 * (1 − 1 / (γ u)^2)) / (1 + γ u)^2 + (2 * γ u) / (γ u * (1 +
γ u))
    using norm-square-gamma-factor[OF assms]
    by presburger
  also have . . . = ((γ u)^2 − 1) / (1 + γ u)^2 + (2 * γ u) / (γ u * (1 + γ u))
    using ‹γ u ≠ 0›
    by (simp add: right-diff-distrib)
  also have . . . = (γ u * ((γ u)^2 − 1)) / (γ u * (1 + γ u)^2) + (2 * γ u * (1
+ γ u)) / (γ u * (1 + γ u)^2)
    using ‹γ u ≠ 0›
    by (simp add: mult.commute power2-eq-square)
  also have . . . = (γ u * ((γ u)^2 − 1) + 2 * γ u * (1 + γ u)) / (γ u * (1 + γ
u)^2)
    by argo
  also have . . . = ((γ u)^3 + γ u + 2 * (γ u)^2) / (γ u * (1 + γ u)^2)
   by (simp add: power2-eq-square power3-eq-cube right-diff-distrib ring-class.ring-distribs(1))
  also have . . . = (γ u * (1 + γ u) ^ 2) / (γ u * (1 + γ u)^2)
    by (simp add: field-simps power2-eq-square power3-eq-cube)
  finally show ?thesis
    using *
    by simp
qed


end
theory PoincareDisc
  imports Complex-Main HOL−Analysis.Inner-Product GammaFactor
begin

typedef PoincareDisc = {z::complex. cmod z < 1}
  morphisms to-complex of-complex
```

by (*rule exI* [**where** *x=0*], *auto*)

setup-lifting *type-definition-PoincareDisc*

lemma *poincare-disc-two-elems*:
shows $\exists$ *z1 z2::PoincareDisc. z1* $\neq$ *z2*
proof−
have *cmod 0* < *1*
by *simp*
moreover
have *cmod* (*1/2*) < *1*
by *simp*
moreover
have (*0::complex*) $\neq$ *1/2*
by *simp*
ultimately
show *?thesis*
by *transfer blast*
qed

lift-definition *inner-p* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *real* (**infixl** · *100*) **is** *inner* **.**

lift-definition *norm-p* :: *PoincareDisc* ⇒ *real* (⟪-⟫ [*100*] *101*) **is** *norm* **.**

lemma *norm-lt-one*:
shows ⟪*u*⟫ < *1*
by *transfer simp*

lemma *norm-geq-zero*:
shows ⟪*u*⟫ ≥ *0*
by *transfer simp*

lemma *square-norm-inner*:
shows (⟪*u*⟫)^2 = *u* · *u*
by *transfer* (*simp add*: *dot-square-norm*)

lift-definition *gammma-factor-p* :: *PoincareDisc* ⇒ *real* (γ_p) **is** *gamma-factor* **.**

lemma *gamma-factor-p-nonzero* [*simp*]:
shows γ_p *u* $\neq$ *0*
apply *transfer*
unfolding *gamma-factor-def*
using *gamma-factor-nonzero*
by *auto*

lemma *gamma-factor-p-positive* [*simp*]:
shows γ_p *u* > *0*
by *transfer* (*simp add*: *gamma-factor-positive*)

lemma *norm-square-gamma-factor-p*:
shows $(\langle\!\langle u\rangle\!\rangle)\hat{}2 = 1 - 1 / (\gamma_p\ u)\hat{}2$
by *transfer* (*simp add*: *norm-square-gamma-factor*)

lemma *norm-square-gamma-factor-p'*:
shows $(\langle\!\langle u\rangle\!\rangle)\hat{}2 = ((\gamma_p\ u)\hat{}2 - 1) / (\gamma_p\ u)\hat{}2$
by *transfer* (*simp add*: *norm-square-gamma-factor'*)

lemma *gamma-factor-p-square-norm*:
shows $(\gamma_p\ u)^2 = 1 / (1 - (\langle\!\langle u\rangle\!\rangle)^2)$
by *transfer* (*simp add*: *gamma-factor-square-norm*)

end
theory *MobiusGyroGroup*
imports *Complex-Main HOL.Real-Vector-Spaces HOL.Transcendental MoreComplex*
GyroGroup PoincareDisc
begin

definition *ozero-m'* :: *complex* **where**
ozero-m' = *0*

lift-definition *ozero-m* :: *PoincareDisc* (0_m) **is** *ozero-m'*
unfolding *ozero-m'-def*
by *simp*

lemma *to-complex-0* [*simp*]:
shows *to-complex* $0_m = 0$
by *transfer* (*simp add*: *ozero-m'-def*)

lemma *to-complex-0-iff* [*iff*]:
shows *to-complex* $x = 0 \longleftrightarrow x = 0_m$
by *transfer* (*simp add*: *ozero-m'-def*)

definition *oplus-m'* :: *complex* ⇒ *complex* ⇒ *complex* **where**
oplus-m' a z = (*a* + *z*) / (*1* + (*cnj a*) ∗ *z*)

lemma *oplus-m'-in-disc*:
assumes *cmod c1* < *1 cmod c2* < *1*
shows *cmod* (*oplus-m' c1 c2*) < *1*
proof−
have *Im* ((*c1* + *c2*) ∗ (*cnj c1* + *cnj c2*)) = *0*
by (*metis complex-In-mult-cnj-zero complex-cnj-add*)
moreover
have *Im* ((*1* + *cnj c1* ∗ *c2*) ∗ (*1* + *c1* ∗ *cnj c2*)) = *0*
by (*cases c1*, *cases c2*, *simp add*: *field-simps*)
ultimately
have *1*: *Re* (*oplus-m' c1 c2* ∗ *cnj* (*oplus-m' c1 c2*)) =

```
      Re (((c1 + c2) * (cnj c1 + cnj c2))) /
      Re (((1 + cnj c1 * c2) * (1 + c1 * cnj c2)))
    unfolding oplus-m'-def
    by (simp add: complex-is-Real-iff)

  have Re (((c1 + c2) * (cnj c1 + cnj c2))) =
      (cmod c1)^2 + (cmod c2)^2 + Re (cnj c1 * c2 + c1 * cnj c2)
    by (smt Re-complex-of-real complex-norm-square plus-complex.simps(1) semir-
ing-normalization-rules(34) semiring-normalization-rules(7))
  moreover
  have Re (((1 + cnj c1 * c2) * (1 + c1 * cnj c2))) =
       Re (1 + cnj c1 * c2 + cnj c2 * c1 + c1 * cnj c1 * c2 * cnj c2)
    by (simp add: field-simps)
  hence *: Re (((1 + cnj c1 * c2) * (1 + c1 * cnj c2))) =
       1 + Re (cnj c1 * c2 + c1 * cnj c2) + (cmod c1)^2 * (cmod c2)^2
   by (smt Re-complex-of-real ab-semigroup-mult-class.mult-ac(1) complex-In-mult-cnj-zero
complex-cnj-one complex-norm-square one-complex.simps(1) one-power2 plus-complex.simps(1)
power2-eq-square semiring-normalization-rules(7) times-complex.simps(1))
  moreover
  have (cmod c1)^2 + (cmod c2)^2 < 1 + (cmod c1)^2 * (cmod c2)^2
  proof-
    have (cmod c1)^2 < 1 (cmod c2)^2 < 1
      using assms
      by (simp-all add: cmod-def)
    hence (1 - (cmod c1)^2) * (1 - (cmod c2)^2) > 0
      by simp
    thus ?thesis
      by (simp add: field-simps)
  qed
  ultimately
  have Re (((c1 + c2) * (cnj c1 + cnj c2))) < Re (((1 + cnj c1 * c2) * (1 + c1
* cnj c2)))
    by simp
  moreover
  have Re (((1 + cnj c1 * c2) * (1 + c1 * cnj c2))) > 0
   by (smt * Re-complex-div-lt-0 calculation complex-cnj-add divide-self mult-zero-left
one-complex.simps(1) zero-complex.simps(1))
  ultimately
  have 2: Re (((c1 + c2) * (cnj c1 + cnj c2))) / Re (((1 + cnj c1 * c2) * (1 +
c1 * cnj c2))) < 1
    by (simp add: divide-less-eq)

  have Re (oplus-m' c1 c2 * cnj (oplus-m' c1 c2)) < 1
    using 1 2
    by simp

  thus ?thesis
    by (simp add: complex-mod-sqrt-Re-mult-cnj)
qed
```

```
lift-definition oplus-m :: PoincareDisc ⇒ PoincareDisc ⇒ PoincareDisc (infixl
⊕m 100) is oplus-m'
proof−
  fix c1 c2
  assume cmod c1 < 1 cmod c2 < 1
  thus cmod (oplus-m' c1 c2) < 1
    by (simp add: oplus-m'-in-disc)
qed

definition ominus-m' :: complex ⇒ complex where
  ominus-m' z = − z

lemma ominus-m'-in-disc:
  assumes cmod z < 1
  shows cmod (ominus-m' z) < 1
  using assms
  unfolding ominus-m'-def
  by simp

lift-definition ominus-m :: PoincareDisc ⇒ PoincareDisc (⊖m) is ominus-m'
proof−
  fix c
  assume cmod c < 1
  thus cmod (ominus-m' c) < 1
    by (simp add: ominus-m'-def)
qed

lemma m-left-id:
  shows 0m ⊕m a = a
  by (transfer, simp add: oplus-m'-def ozero-m'-def)

lemma m-left-inv:
  shows ⊖m a ⊕m a = 0m
  by (transfer, simp add: oplus-m'-def ominus-m'-def ozero-m'-def)

definition gyr-m' :: complex ⇒ complex ⇒ complex ⇒ complex where
  gyr-m' a b z = ((1 + a ∗ cnj b) / (1 + cnj a ∗ b)) ∗ z

lift-definition gyrm :: PoincareDisc ⇒ PoincareDisc ⇒ PoincareDisc ⇒ Poincar-
eDisc is gyr-m'
proof−
  fix a b z
  assume cmod a < 1 cmod b < 1 cmod z < 1
  have cmod (1 + a ∗ cnj b) = cmod (1 + cnj a ∗ b)
    by (metis complex-cnj-add complex-cnj-cnj complex-cnj-mult complex-cnj-one
complex-mod-cnj)
  hence cmod ((1 + a ∗ cnj b) / (1 + cnj a ∗ b)) = 1
    by (simp add: ‹cmod a < 1› ‹cmod b < 1› den-not-zero norm-divide)
```

thus *cmod* (*gyr-m′ a b z*) < *1*
using ‹*cmod z* < *1*›
unfolding *gyr-m′-def*
by (*metis mult-cancel-right1 norm-mult*)
qed

lemma *gyr-m-commute*:
$a \oplus_m b = gyr_m\ a\ b\ (b \oplus_m a)$
by *transfer* (*metis* (*no-types*, *opaque-lifting*) *oplus-m′-def gyr-m′-def add.commute den-not-zero mult.commute nonzero-mult-divide-mult-cancel-right2 times-divide-times-eq*)

lemma *gyr-m-left-assoc*:
$a \oplus_m (b \oplus_m z) = (a \oplus_m b) \oplus_m gyr_m\ a\ b\ z$
proof *transfer*
fix *a b z*
assume ∗: *cmod a* < *1 cmod b* < *1 cmod z* < *1*
have *1*: *oplus-m′ a* (*oplus-m′ b z*) =
(*a* + *b* + (*1* + *a* ∗ *cnj b*) ∗ *z*) /
((*cnj a* + *cnj b*) ∗ *z* + (*1* + *cnj a* ∗ *b*))
unfolding *gyr-m′-def oplus-m′-def*
by (*smt* ∗(*2*) ∗(*3*) *ab-semigroup-mult-class.mult-ac*(*1*) *add.left-commute add-divide-eq-iff combine-common-factor den-not-zero divide-divide-eq-right mult.commute mult-cancel-right2 nonzero-mult-div-cancel-left semiring-normalization-rules*(*1*) *semiring-normalization-rules*(*23*) *semiring-normalization-rules*(*34*) *times-divide-eq-right*)
have *2*: *oplus-m′* (*oplus-m′ a b*) (*gyr-m′ a b z*) =
((*a* + *b*) + (*1* + *a* ∗ *cnj b*) ∗ *z*) /
((*cnj a* + *cnj b*) ∗ *z* + (*1* + *cnj a* ∗ *b*))
proof−
have *x*: ((*a* + *b*) / (*1* + *cnj a* ∗ *b*) +
(*1* + *a* ∗ *cnj b*) / (*1* + *cnj a* ∗ *b*) ∗ *z*) =
((*a* + *b*) + (*1* + *a* ∗ *cnj b*) ∗ *z*) / (*1* + *cnj a* ∗ *b*)
by (*metis add-divide-distrib times-divide-eq-left*)
moreover
have *1* + *cnj* ((*a* + *b*) / (*1* + *cnj a* ∗ *b*)) ∗
((*1* + *a* ∗ *cnj b*) / (*1* + *cnj a* ∗ *b*) ∗ *z*) =
1 + (*cnj a* + *cnj b*) / (*1* + *cnj a* ∗ *b*) ∗ *z*
using *divide-divide-times-eq divide-eq-0-iff mult-eq-0-iff nonzero-mult-div-cancel-left*
by *force*
hence *y*: *1* + *cnj* ((*a* + *b*) / (*1* + *cnj a* ∗ *b*)) ∗
((*1* + *a* ∗ *cnj b*) / (*1* + *cnj a* ∗ *b*) ∗ *z*) =
((*cnj a* + *cnj b*) ∗ *z* + (*1* + *cnj a* ∗ *b*)) / (*1* + *cnj a* ∗ *b*)
by (*metis* ∗(*1*) ∗(*2*) *add.commute add-divide-distrib den-not-zero divide-self times-divide-eq-left*)
ultimately
show *?thesis*
unfolding *gyr-m′-def oplus-m′-def*
by (*subst x*, *subst y*, *simp add*: ∗(*1*) ∗(*2*) *den-not-zero*)
qed
show *oplus-m′ a* (*oplus-m′ b z*) =

oplus-m′ (*oplus-m′ a b*) (*gyr-m′ a b z*)
by (*subst 1*, *subst 2*, *simp*)
qed

lemma *gyr-m-inv*:
gyr_m *a b* (gyr_m *b a z*) = *z*
by *transfer* (*simp add*: *gyr-m′-def*, *metis den-not-zero nonzero-mult-div-cancel-left nonzero-mult-divide-mult-cancel-right semiring-normalization-rules*(*7*))

lemma *gyr-m-bij*:
shows *bij* (gyr_m *a b*)
by (*metis bij-betw-def inj-def gyr-m-inv surj-def*)

lemma *gyr-m-not-degenerate*:
shows ∃ *z1 z2*. gyr_m *a b z1* ≠ gyr_m *a b z2*
proof−
obtain *z1 z2* :: *PoincareDisc* **where** *z1* ≠ *z2*
using *poincare-disc-two-elems*
by *blast*
hence gyr_m *a b z1* ≠ gyr_m *a b z2*
by (*metis gyr-m-inv*)
thus *?thesis*
by *blast*
qed

lemma *gyr-m-left-loop*:
shows gyr_m *a b* = gyr_m (*a* $\oplus_m$ *b*) *b*
proof−
have ∃ *z*. gyr_m (*a* $\oplus_m$ *b*) *b z* ≠ 0_m
using *gyr-m-not-degenerate*
by *metis*
hence ⋀ *z*. gyr_m *a b z* = gyr_m (*a* $\oplus_m$ *b*) *b z*
proof *transfer*
fix *a b z*
assume ∃*z*∈{*z*. *cmod z* < *1*}. *gyr-m′* (*oplus-m′ a b*) *b z* ≠ *ozero-m′*
then obtain *z′* **where**
cmod z′ < *1 gyr-m′* (*oplus-m′ a b*) *b z′* ≠ *ozero-m′*
by *auto*
hence ∗: *1* + (*a* + *b*) / (*1* + *cnj a* ∗ *b*) ∗ *cnj b* ≠ *0*
by (*simp add*: *gyr-m′-def oplus-m′-def ozero-m′-def*)
assume *cmod a* < *1 cmod b* < *1 cmod z* < *1*
have *1*: *1* + (*a* + *b*) / (*1* + *cnj a* ∗ *b*) ∗ *cnj b* =
(*1* + *cnj a* ∗ *b* + *a* ∗ *cnj b* + *b* ∗ *cnj b*) / (*1* + *cnj a* ∗ *b*)
using ‹*cmod a* < *1*› ‹*cmod b* < *1*› *add-divide-distrib den-not-zero divide-self times-divide-eq-left*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-add-class.add-ac*(*1*) *distrib-right*)
have *2*: *1* + *cnj* ((*a* + *b*) / (*1* + *cnj a* ∗ *b*)) ∗ *b* =
(*1* + *cnj a* ∗ *b* + *a* ∗ *cnj b* + *b* ∗ *cnj b*) / (*1* + *a* ∗ *cnj b*)
by (*smt 1 complex-cnj-add complex-cnj-cnj complex-cnj-divide complex-cnj-mult*

```
complex-cnj-one semiring-normalization-rules(23) semiring-normalization-rules(7))
    have 1 + cnj a * b + a * cnj b + b * cnj b ≠ 0
      using * 1
      by auto
    then show gyr-m' a b z = gyr-m' (oplus-m' a b) b z
      unfolding gyr-m'-def oplus-m'-def
      by (subst 1, subst 2, simp)
  qed
  thus ?thesis
    by auto
qed

lemma gyr-m-distrib:
  shows gyr_m a b (a' ⊕_m b') = gyr_m a b a' ⊕_m gyr_m a b b'
  apply transfer
  apply (simp add: gyr-m'-def oplus-m'-def)
  apply (simp add: add-divide-distrib distrib-left)
  done

interpretation Mobius-gyrogroup: gyrogroup ozero-m oplus-m ominus-m gyr_m
proof
  fix a
  show 0_m ⊕_m a = a
    by (simp add: m-left-id)
next
  fix a
  show ⊖_m a ⊕_m a = 0_m
    by (simp add: m-left-inv)
next
  fix a b z
  show a ⊕_m (b ⊕_m z) = a ⊕_m b ⊕_m gyr_m a b z
    by (simp add: gyr-m-left-assoc)
next
  fix a b
  show gyr_m a b = gyr_m (a ⊕_m b) b
    using gyr-m-left-loop by auto
next
  fix a b
  show gyrogroupoid.gyroaut (⊕_m) (gyr_m a b)
    unfolding gyrogroupoid.gyroaut-def
  proof safe
    fix a' b'
    show gyr_m a b (a' ⊕_m b') = gyr_m a b a' ⊕_m gyr_m a b b'
      by (simp add: gyr-m-distrib)
  next
    show bij (gyr_m a b)
      by (simp add: gyr-m-bij)
  qed
qed
```

```
interpretation Mobius-gyrocommutative-gyrogroup: gyrocommutative-gyrogroup ozero-m
oplus-m ominus-m gyr_m
proof
  fix a b
  show a ⊕_m b = gyr_m a b (b ⊕_m a)
    using gyr-m-commute by blast
qed

instantiation PoincareDisc :: gyrogroupoid
begin
definition gyrozero-PoincareDisc where
 gyrozero-PoincareDisc = ozero-m
definition gyroplus-PoincareDisc where
 gyroplus-PoincareDisc = oplus-m
instance ..
end

instantiation PoincareDisc :: gyrogroup
begin
definition gyroinv-PoincareDisc where
 gyroinv-PoincareDisc = ominus-m
definition gyr-PoincareDisc where
 gyr-PoincareDisc = gyr_m
instance proof
  fix a :: PoincareDisc
  show 0_g ⊕ a = a
    by (simp add: gyroplus-PoincareDisc-def gyrozero-PoincareDisc-def)
next
  fix a :: PoincareDisc
  show ⊖ a ⊕ a = 0_g
   by (simp add: gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def gyrozero-PoincareDisc-def)
next
  fix a b :: PoincareDisc
  show gyroaut (gyr a b)
   by (simp add: gyr-PoincareDisc-def gyroaut-def gyroplus-PoincareDisc-def gyr-m-bij)
next
  fix a b z :: PoincareDisc
  show a ⊕ (b ⊕ z) = a ⊕ b ⊕ gyr a b z
   by (simp add: gyr-PoincareDisc-def gyroplus-PoincareDisc-def gyr-m-left-assoc)
next
  fix a b :: PoincareDisc
  show  gyr a b = gyr (a ⊕ b) b
    using gyr-PoincareDisc-def gyroplus-PoincareDisc-def gyr-m-left-loop by auto
qed
end

instantiation PoincareDisc :: gyrocommutative-gyrogroup
begin
```

instance proof
  fix *a b* :: *PoincareDisc*
  show $a \oplus b = gyr\ a\ b\ (b \oplus a)$
    using *gyr-PoincareDisc-def gyroplus-PoincareDisc-def gyr-m-commute* **by** *auto*
qed
end

lemma *oplusM-reals*:
  assumes $Im\ (to\text{-}complex\ x) = 0\ Im\ (to\text{-}complex\ y) = 0$
  shows $Im\ (to\text{-}complex\ (x \oplus_m y)) = 0$
  using *assms*
  by (*transfer*, *auto simp add*: *oplus-m′-def complex-is-Real-iff*)

lemma *oplusM-pos-reals*:
  assumes $Im\ (to\text{-}complex\ x) = 0\ Im\ (to\text{-}complex\ y) = 0$
  assumes $Re\ (to\text{-}complex\ x) \geq 0\ Re\ (to\text{-}complex\ y) \geq 0$
  shows $Re\ (to\text{-}complex\ (x \oplus_m y)) \geq 0$
  using *assms*
  by (*transfer*, *auto simp add*: *oplus-m′-def complex-is-Real-iff*)

definition gyr_m*-alternative* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc* **where**
  $gyr_m\text{-}alternative\ u\ v\ w = \ominus_m (u \oplus_m v) \oplus_m (u \oplus_m (v \oplus_m w))$

lemma *gyr-m-alternative-gyr-m*:
  shows $gyr_m\text{-}alternative\ u\ v\ w = gyr_m\ u\ v\ w$
  by (*metis* gyr_m*-alternative-def gyr-m-inv gyr-m-left-assoc gyr-m-left-loop m-left-id m-left-inv*)

definition *oplus-m′-alternative* :: *complex* ⇒ *complex* ⇒ *complex* **where**
  oplus-m′-alternative u v =
    $((1 + 2*inner\ u\ v + (norm\ v)\hat{}2) *_R u + (1 - (norm\ u)\hat{}2) *_R v)$ /
     $(1 + 2*inner\ u\ v + (norm\ u)\hat{}2 * (norm\ v)\hat{}2)$

lemma *oplus-m′-alternative*:
  assumes $cmod\ u < 1\ cmod\ v < 1$
  shows *oplus-m′-alternative u v* = *oplus-m′ u v*
proof−
  have ∗: $2 * inner\ u\ v = cnj\ u * v + cnj\ v * u$
    using *two-inner-cnj*
    by *auto*

  have $(1 + 2*inner\ u\ v + (norm\ v)\hat{}2) * u =$
       $(1 + cnj\ u *v + cnj\ v * u + (norm\ v)\hat{}2) * u$
    using ∗
    by *auto*

moreover

have *1 + 2∗inner u v + (norm u)^2 ∗ (norm v)^2 =*
1 + cnj u ∗ v + cnj v ∗ u + (norm u)^2 ∗ (norm v)^2
using ∗
by *auto*

moreover

have *(1 + cnj u ∗ v + cnj v ∗ u + (norm v)^2) ∗ u + (1 − (norm u)^2) ∗ v =*
(1 + cnj v ∗ u) ∗ (u + v)
proof−
have ∗: *(1 + cnj u ∗ v + cnj v ∗ u + (norm v)^2) ∗ u =*
u + (norm u)^2 ∗ v + cnj v ∗ u^2 + (norm v)^2 ∗ u
by *(smt (verit, del-insts) ab-semigroup-mult-class.mult-ac(1) comm-semiring-class.distrib complex-norm-square mult.commute mult-cancel-right1 power2-eq-square)*
have ∗∗: *(1 + cnj v ∗ u) ∗ (u + v) = u + cnj v ∗ u ∗ u + v + cnj v ∗ u ∗ v*
by *(simp add: distrib-left ring-class.ring-distribs(2))*
have *u + cnj u ∗ v ∗u + v + cnj u∗ v ∗ v = u + cnj u ∗ v^2 + (norm u)^2 ∗ v + v*
by *(simp add: cnj-cmod mult.commute power2-eq-square)*
have ∗∗∗: *(1 − (norm u)^2) ∗ v = v − (norm u)^2 ∗ v*
by *(simp add: mult.commute right-diff-distrib′)*
have *(1 + cnj u ∗ v + cnj v ∗ u + (norm v)^2) ∗ u + (1 − (norm u)^2) ∗ v =*
u + (norm u)^2 ∗ v + (cnj v) ∗ u^2 + (norm v)^2 ∗ u + v − (norm u)^2 ∗ v
using ∗ ∗∗∗
by *force*
have ∗∗∗∗: *(1 + cnj u ∗ v + cnj v ∗ u + (norm v)^2) ∗ u + (1−(norm u)^2) ∗ v =*
u + cnj v ∗u^2 + (norm v)^2 ∗ u + v
using ∗ ∗∗∗
by *auto*

have *(1 + cnj v ∗ u) ∗ (u+v) = u + (norm v)^2 ∗u + v + cnj v ∗ u^2*
using ∗∗
by *(simp add: cnj-cmod mult.commute power2-eq-square)*

then show *?thesis*
using ∗∗∗∗
by *auto*
qed

moreover have *1 + cnj u ∗ v + cnj v ∗u + (norm u)^2 ∗ (norm v)^2 =*
(1 + cnj u ∗ v) ∗ (1 + cnj v ∗ u)
by *(smt (verit, del-insts) cnj-cmod comm-semiring-class.distrib complex-cnj-cnj complex-cnj-mult complex-mod-cnj is-num-normalize(1) mult.commute mult-numeral-1 norm-mult numeral-One power-mult-distrib)*

ultimately
show *?thesis*
using *assms*
unfolding *oplus-m′-alternative-def oplus-m′-def*
by (*metis* (*no-types*, *lifting*) *den-not-zero divide-divide-eq-left′ nonzero-mult-div-cancel-left scaleR-conv-of-real*)
qed

lift-definition *oplus-m-alternative* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc* **is** *oplus-m′-alternative*
by (*simp add*: *oplus-m′-alternative oplus-m′-in-disc*)

end
theory *Gyrotrigonometry*
imports *Main GyroVectorSpace*
begin

datatype ′*a otriangle* = *M-gyrotriangle* (*A*:′*a*) (*B*:′*a*) (*C*:′*a*)

context *pre-gyrovector-space*
begin

definition *unit* :: ′*a* ⇒ ′*b* **where**
unit a = *to-carrier a* $/_R$ ⟪*a*⟫

lemma *norm-inner-le-1*:
fixes *a b* :: ′*b*
assumes *norm a* ≤ *1* *norm b* ≤ *1*
shows *norm* (*inner a b*) ≤ *1*
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *Cauchy-Schwarz-ineq2 mult-le-one norm-ge-zero real-norm-def*)

lemma *norm-inner-unit*:
shows *norm* (*inner* (*unit* (⊖ *a* ⊕ *b*)) (*unit* (⊖ *a* ⊕ *c*))) ≤ *1*
proof−
have *norm* (*unit* (⊖ *a* ⊕ *b*)) = ⟪⊖ *a* ⊕ *b*⟫ / ⟪⊖ *a* ⊕ *b*⟫
by (*simp add*: *unit-def gyronorm-def*)
then have *norm* (*unit* (⊖ *a* ⊕ *b*)) ≤ *1*
by *simp*
moreover
have *norm* (*unit* (⊖ *a* ⊕ *c*)) = ⟪⊖ *a* ⊕ *c*⟫ / ⟪⊖ *a* ⊕ *c*⟫
by (*simp add*: *unit-def gyronorm-def*)
then have *norm* (*unit* (⊖ *a* ⊕ *c*)) ≤ *1*
by *simp*
ultimately
show *?thesis*
using *norm-inner-le-1*
by *blast*

qed

definition *angle* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow real$ **where**
  angle a b c $=$ *arccos* (*inner* (*unit* ($\ominus$ *a* $\oplus$ *b*)) (*unit* ($\ominus$ *a* $\oplus$ *c*)))

definition *o-ray* :: $'a \Rightarrow 'a \Rightarrow 'a$ *set* **where**
  o-ray x p $= \{s{::}'a.\ \exists\, t{::}real.\ t \geq 0 \wedge s = (x \oplus t \otimes (\ominus x \oplus p))\}$

lemma *T8-5*:
  assumes *b2* $\in$ *o-ray a1 b1 b2* $\neq$ *a1*
      c2 $\in$ *o-ray a1 c1 c2* $\neq$ *a1*
  shows *angle a1 b1 c1* $=$ *angle a1 b2 c2*
proof−
  obtain *t1*::*real* **where** *t1*: *t1* $>$ *0 b2* $=$ *a1* $\oplus$ *t1* $\otimes$ ($\ominus$ *a1* $\oplus$ *b1*)
    using *assms less-eq-real-def o-ray-def* **by** *auto*
  then have $\ominus$ *a1* $\oplus$ *b2* $=$ *t1* $\otimes$ ($\ominus$ *a1* $\oplus$ *b1*)
     using *gyro-left-cancel′* **by** *simp*

  obtain *t2*::*real* **where** *t2*: *t2* $>$ *0 c2* $=$ *a1* $\oplus$ *t2* $\otimes$ ($\ominus$ *a1* $\oplus$ *c1*)
    using *assms less-eq-real-def o-ray-def* **by** *auto*
  then have $\ominus$ *a1* $\oplus$ *c2* $=$ *t2* $\otimes$ ($\ominus$ *a1* $\oplus$ *c1*)
    using *gyro-left-cancel′* **by** *simp*

  have *angle a1 b2 c2* $=$ *arccos* (*inner* (*to-carrier* ($\ominus$ *a1* $\oplus$ *b2*) $/_R$ $\langle\!\langle\ominus$ *a1* $\oplus$ *b2*$\rangle\!\rangle$)
                (*to-carrier* ($\ominus$ *a1* $\oplus$ *c2*) $/_R$ $\langle\!\langle\ominus$ *a1* $\oplus$ *c2*$\rangle\!\rangle$))
    using *angle-def unit-def* **by** *presburger*
  also have ... $=$
        arccos (*inner* (*to-carrier* (*t1* $\otimes$ ($\ominus$ *a1* $\oplus$ *b1*)) $/_R$ $\langle\!\langle$*t1* $\otimes$ ($\ominus$ *a1* $\oplus$ *b1*)$\rangle\!\rangle$)
             (*to-carrier* (*t2* $\otimes$ ($\ominus$ *a1* $\oplus$ *c1*)) $/_R$ $\langle\!\langle$*t2* $\otimes$ ($\ominus$ *a1* $\oplus$ *c1*)$\rangle\!\rangle$))
    using ‹$\ominus$ *a1* $\oplus$ *b2* $=$ *t1* $\otimes$ ($\ominus$ *a1* $\oplus$ *b1*)› ‹$\ominus$ *a1* $\oplus$ *c2* $=$ *t2* $\otimes$ ($\ominus$ *a1* $\oplus$ *c1*)›
by *auto*
  finally show *?thesis*
    unfolding *angle-def unit-def*
    using *t1 t2*
    by (*smt* (*verit*, *best*) *scale-prop1 times-zero*)
qed

definition *get-a* :: *'a otriangle* $\Rightarrow$ *'a* **where**
  get-a t $= \ominus$ (*C t*) $\oplus$ (*B t*)
definition *get-b* :: *'a otriangle* $\Rightarrow$ *'a* **where**
  get-b t $= \ominus$ (*C t*) $\oplus$ (*A t*)
definition *get-c* :: *'a otriangle* $\Rightarrow$ *'a* **where**
  get-c t $= \ominus$ (*B t*) $\oplus$ (*A t*)

definition *get-alpha* :: *'a otriangle* $\Rightarrow$ *real* **where**
  get-alpha t $=$ *angle* (*A t*) (*B t*) (*C t*)
definition *get-beta* :: *'a otriangle* $\Rightarrow$ *real* **where**
  get-beta t $=$ *angle* (*B t*) (*C t*) (*A t*)
definition *get-gamma* :: *'a otriangle* $\Rightarrow$ *real* **where**

```
  get-gamma t = angle (C t) (A t) (B t)

definition cong-gyrotriangles :: 'a otriangle ⇒ 'a otriangle ⇒ bool where
  cong-gyrotriangles t1 t2 ⟷
     (⟪get-a t1⟫ = ⟪get-a t2⟫ ∧ ⟪get-b t1⟫ = ⟪get-b t2⟫ ∧ ⟪get-c t1⟫ = ⟪get-c t2⟫
∧
      (get-alpha t1 = get-alpha t2) ∧ (get-beta t1 = get-beta t2) ∧ (get-gamma t1
= get-gamma t2))
end

end
theory HyperbolicFunctions
imports HOL.Transcendental
begin

lemma artanh-abs-tanh:
  fixes x::real
  shows artanh (abs (tanh x)) = abs x
proof (cases x ≥ 0)
  case True
  then show ?thesis
    by (simp add: artanh-tanh-real)
next
  case False
  then show ?thesis
    by (metis artanh-tanh-real tanh-real-abs)
qed

lemma artanh-nonneg:
  fixes x :: real
  assumes 0 ≤ x x < 1
  shows artanh x ≥ 0
proof−
  have (1+x)/(1−x) ≥ 1/(1−x)
   by (metis assms add-0 add-increasing2 divide-right-mono le-diff-eq less-eq-real-def)
  moreover have 1/(1−x) ≥ 1
    using assms
    by simp
  moreover have artanh x = 1/2∗ln((1+x)/(1−x))
    by (simp add: artanh-def)
  moreover have ln((1+x)/(1−x))≥0
    using calculation(1) calculation(2) by fastforce
  moreover have ((artanh x)≥0)
    using calculation(3) calculation(4) by linarith
  moreover have (0≤x ∧ x<1)⟶ ((artanh x)≥0)
    using calculation by blast
  ultimately
  show ?thesis
    by blast
```

```
qed

lemma artanh-not-0:
  fixes x :: real
  assumes x > 0 x < 1
  shows artanh x ≠ 0
  using assms
  by (simp add: artanh-def)

lemma tanh-not-0:
  fixes x :: real
  assumes x > 0 x < 1
  shows tanh x ≠ 0
  using assms
  by simp

lemma tanh-monotone:
  fixes x y :: real
  assumes x > y
  shows tanh x > tanh y
  using assms
  by simp

lemma artanh-monotone1:
  fixes x::real
  assumes x ≥ 0 x < 1 y ≥ 0 y < 1 x ≤ y
  shows (1+x) / (1−x) ≤ (1+y) / (1−y)
  using assms
  by (smt (verit, best) frac-less)

lemma artanh-monotone2:
  fixes x::real
  assumes x≥0 x<1 y≥0 y<1 x≤y
  shows ln ((1+x)/(1−x)) ≤ ln((1+y)/(1−y))
  using artanh-monotone1 assms(1) assms(4) assms(5) by force

lemma artanh-monotone:
  fixes x y :: real
  assumes x ≥ 0 x < 1 0 ≤ y y < 1
  assumes x ≤ y
  shows artanh x ≤ artanh y
proof−
  have artanh x = 1/2 ∗ ln((1+x)/(1−x))
    by (simp add: artanh-def)
  moreover have artanh y = (1/2) ∗ ln((1+y)/(1−y))
    by (simp add: artanh-def)
  ultimately show ?thesis
    using assms artanh-monotone2
    by (simp add: artanh-def)
```

qed

lemma *tanh-artanh-nonneg*:
  fixes *x r* :: *real*
  assumes $r \geq 0$ $x \geq 0$ $x < 1$
  shows *tanh* (*r* ∗ *artanh x*) ≥ *0*
  using *assms*
  by (*simp add*: *artanh-nonneg*)

lemma *tanh-artanh-mono*:
  fixes *x y* :: *real*
  assumes $0 \leq x$ $x < 1$ $0 \leq y$ $y < 1$
  assumes $x \leq y$
  shows *tanh* (*2* ∗ *artanh x*) ≤ *tanh* (*2* ∗ *artanh y*)
  using *assms*
  using *artanh-monotone*
  by *auto*

lemma *tanh-def′*:
  fixes *x* :: *real*
  shows *tanh x* = (*exp* (*2*∗*x*) − *1*) / (*exp* (*2*∗*x*) + *1*)
  unfolding *tanh-def sinh-def cosh-def*
  by (*metis cosh-def exp-gt-zero exp-of-nat-mult ln-unique of-nat-numeral sinh-def tanh-def tanh-ln-real*)

lemma *tanh-artanh*:
  fixes *x* :: *real*
  assumes $-1 < x$ $x < 1$
  shows *tanh* (*artanh x*) = *x*
  using *assms*
  unfolding *artanh-def tanh-def′*
  by (*simp add*: *field-simps*)

end
theory *MobiusGyroVectorSpace*
imports *Main MobiusGyroGroup GyroVectorSpace Gyrotrigonometry GammaFactor HyperbolicFunctions*
begin

lemma *norms*:
  shows {*x*. ∃ *a*. *x* = *cmod* (*to-complex a*)} ∪ {*x*. ∃ *a*. *x* = − *cmod* (*to-complex a*)} = {*x*. |*x*| < *1*}
proof−
  {
    fix *x* :: *real*
    assume |*x*| < *1*
    then have *cmod* (*Complex x 0*) < *1*

```
      by (simp add: complex-norm)
    then have to-complex (PoincareDisc.of-complex (Complex x 0)) = x
      by (simp add: complex-of-real-def of-complex-inverse)
    then have ∃ a. x = cmod (to-complex a) ∨ (∃ a. x = − cmod (to-complex a))
      by (metis abs-eq-iff′ norm-of-real)
  }
  moreover
  have ⋀ a. cmod (to-complex a) < 1
    using to-complex by force
  ultimately show ?thesis
    by auto
qed

global-interpretation Mobius-gyrocarrier′: gyrocarrier′
  where to-carrier = to-complex
rewrites
  Mobius-gyrocarrier′.gyroinner = inner-p and
  Mobius-gyrocarrier′.gyronorm = norm-p and
  Mobius-gyrocarrier′.carrier = {z. cmod z < 1} and
  Mobius-gyrocarrier′.norms = {x. abs x < 1}
defines
  of-complex = gyrocarrier′.of-carrier to-complex
proof−
  show ∗: gyrocarrier′ to-complex
  proof
    show inj to-complex
      by (simp add: inj-on-def to-complex-inject)
  next
    show to-complex 0_g = 0
      by (simp add: gyrozero-PoincareDisc-def ozero-m′-def ozero-m.rep-eq)
  qed

  show gyrocarrier′.gyroinner to-complex = (·)
    apply rule
    apply rule
    unfolding gyrocarrier′.gyroinner-def[OF ∗]
    apply transfer
    by (simp add: inner-complex-def)

  show gyrocarrier′.gyronorm to-complex = norm-p
    apply rule
    unfolding gyrocarrier′.gyronorm-def[OF ∗]
    apply transfer
    by simp

  show gyrocarrier′.carrier to-complex = {z. cmod z < 1}
    unfolding gyrocarrier′.carrier-def[OF ∗]
    using type-definition.Rep-range type-definition-PoincareDisc
    by blast
```

show *gyrocarrier'.norms to-complex* = {*x*. |*x*| < *1*}
using *norms*
unfolding *gyrocarrier'.norms-def*[*OF* ∗]
unfolding *gyrocarrier'.gyronorm-def*[*OF* ∗]
by *auto*
qed

lemma *Mobius-gyrocarrier'-norms* [*simp*]:
shows *gyrocarrier'.norms to-complex* = {*x*. *abs x* < *1*}
using *Mobius-gyrocarrier'.norms-def*
unfolding *gyrocarrier'.norms-def*[*OF Mobius-gyrocarrier'.gyrocarrier'-axioms*]
gyrocarrier'.gyronorm-def[*OF Mobius-gyrocarrier'.gyrocarrier'-axioms*]
using *norm-p.rep-eq*
by *presburger*

lemma *Mobius-gyrocarrier'-carrier* [*simp*]:
shows *gyrocarrier'.carrier to-complex* = {*z*. *cmod z* < *1*}
unfolding *gyrocarrier'.carrier-def*[*OF Mobius-gyrocarrier'.gyrocarrier'-axioms*]
using *type-definition.Rep-range type-definition-PoincareDisc*
by *blast*

lemma *moebius-gyroauto*:
shows gyr_m *u v a* · gyr_m *u v b* = *a* · *b*
proof−
have gyr_m *u v a* · gyr_m *u v b* = *Re*((*cnj* (*to-complex* (gyr_m *u v a*))) ∗ (*to-complex* (gyr_m *u v b*)))
using *inner-p.rep-eq*
by (*simp add*: *inner-complex-def*)
moreover have gyr_m *u v a* = *of-complex*(((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*cnj* (*to-complex u*)) ∗ *to-complex v*)) ∗
(*to-complex a*))
by (*metis Mobius-gyrocarrier'.of-carrier gyr-m'-def* gyr_m*.rep-eq*)
moreover have gyr_m *u v b* = *of-complex*(((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*cnj* (*to-complex u*)) ∗ *to-complex v*)) ∗
(*to-complex b*))
by (*metis Mobius-gyrocarrier'.of-carrier gyr-m'-def* gyr_m*.rep-eq*)
moreover have (*cnj* (*to-complex* (gyr_m *u v a*))) = *cnj* ((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*cnj* (*to-complex u*)) ∗ *to-complex v*)) ∗
cnj (*to-complex a*)
by (*simp add*: *gyr-m'-def* gyr_m*.rep-eq*)
moreover have (*cnj* ((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*to-complex v*)∗(*cnj* (*to-complex u*)))))∗ ((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*to-complex v*)∗(*cnj* (*to-complex u*)))) = *1*
proof−
have ∗: *cmod* (*to-complex u*) < *1*
using *to-complex* **by** *blast*

moreover have ∗∗:*cmod* (*to-complex v*) < *1*
using *to-complex* **by** *blast*
moreover have *cmod* (((*1* + (*to-complex u*) ∗ *cnj* (*to-complex v*)) / (*1* + (*to-complex v*)∗(*cnj* (*to-complex u*))))) =*1*
using *cmod-mix-cnj*[*OF* ∗ ∗∗]
by *force*
ultimately show *?thesis* **using** *cnj-cmod-1*
by (*metis mult.commute*)
qed
moreover have gyr_m *u v a* · gyr_m *u v b* = *Re*((*cnj* (*to-complex a*))∗(*to-complex b*))
using *calculation*(*1*) *calculation*(*5*) *gyr-m′-def* gyr_m*.rep-eq* **by** *force*
moreover have *a* · *b* = *Re*((*cnj* (*to-complex a*))∗(*to-complex b*))
by (*simp add*: *inner-complex-def inner-p.rep-eq*)
ultimately show *?thesis*
by *presburger*
qed

interpretation *Mobius-gyrocarrier*: *gyrocarrier*
where *to-carrier* = *to-complex*
proof
fix *u v a b*
have *gyr u v a* · *gyr u v b* = *a* · *b*
by (*simp add*: *gyr-PoincareDisc-def moebius-gyroauto*)
then show *gyrocarrier′.gyroinner to-complex* (*gyr u v a*) (*gyr u v b*) =
gyrocarrier′.gyroinner to-complex a b
using *gyrocarrier′.gyroinner-def*[*OF Mobius-gyrocarrier′.gyrocarrier′-axioms*] *Mobius-gyrocarrier′.gyroinner-def*
by *fastforce*
qed

global-interpretation *Mobius-gyrocarrier-norms-embed′*: *gyrocarrier-norms-embed′*
where *to-carrier* = *to-complex*
rewrites
Mobius-gyrocarrier-norms-embed′.reals = *of-complex* ‘ *cor* ‘ {*x*. *abs x* < *1*}
proof−
show ∗: *gyrocarrier-norms-embed′ to-complex*
by *unfold-locales auto*

show *gyrocarrier-norms-embed′.reals to-complex* = *of-complex* ‘ *cor* ‘ {*x*. |*x*| < *1*}
unfolding *gyrocarrier-norms-embed′.reals-def*[*OF* ∗]
using *Mobius-gyrocarrier′-norms of-complex-def*
by *presburger*
qed

lemma *Mobius-gyrocarrier-norms-embed′-to-real′*:
assumes *x* ∈ *Mobius-gyrocarrier-norms-embed′.reals*

shows *Mobius-gyrocarrier-norms-embed′.to-real′ x = Re (to-complex x)*
using *assms*
using *Mobius-gyrocarrier-norms-embed′.to-real′-def of-complex-inverse*
by *fastforce*

lemma *Mobius-gyrocarrier-norms-embed′-of-real′*:
assumes *x ∈ Mobius-gyrocarrier′.norms*
shows *Mobius-gyrocarrier-norms-embed′.of-real′ x = PoincareDisc.of-complex (cor x)*
using *assms*
by (*metis Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′.of-real′-def comp-apply mem-Collect-eq norm-of-real of-complex-inverse*)

lemma *gyronorm-Re*:
assumes *Re (to-complex x) ≥ 0 Im (to-complex x) = 0*
shows *⟪x⟫ = Re (to-complex x)*
using *assms*
by (*simp add*: *Mobius-gyrocarrier′.gyronorm-def cmod-eq-Re*)

lemma *Mobius-gyrocarrier-norms-embed′-reals* [*simp*]:
shows *gyrocarrier-norms-embed′.reals to-complex = of-complex ‘ cor ‘ {x. |x| < 1}*
by (*simp add*: *Mobius-gyrocarrier-norms-embed′.gyrocarrier-norms-embed′-axioms gyrocarrier-norms-embed′.reals-def of-complex-def*)

definition *otimes′-k* :: *real ⇒ complex ⇒ real* **where**
otimes′-k r z = ((1 + cmod z) powr r − (1 − cmod z) powr r) /
((1 + cmod z) powr r + (1 − cmod z) powr r)

lemma *otimes′-k-tanh*:
assumes *cmod z < 1*
shows *otimes′-k r z = tanh (r ∗ artanh (cmod z))*
proof−
have *0 < 1 + cmod z*
by (*smt norm-not-less-zero*)
hence *(1 + cmod z) powr r ≠ 0*
by *auto*

have *1 − (1 − cmod z) powr r / (1 + cmod z) powr r =*
((1 + cmod z) powr r − (1 − cmod z) powr r) / (1 + cmod z) powr r
by (*smt ‹(1 + cmod z) powr r ≠ 0› add-divide-distrib divide-self*)
moreover
have *1 + (1 − cmod z) powr r / (1 + cmod z) powr r =*
((1 + cmod z) powr r + (1 − cmod z) powr r) / (1 + cmod z) powr r
by (*smt add-divide-distrib calculation*)
moreover
have *exp (− (r ∗ ln ((1 + cmod z) / (1 − cmod z)))) =*
((1 + cmod z) / (1 − cmod z)) powr (−r)

using ‹$0 < 1 + cmod\ z$› *ln-powr*[*symmetric*, *of* $(1 + cmod\ z) / (1 - cmod\ z)$ $-r$]
using *assms* **by** (*simp add*: *powr-def*)
ultimately
show *?thesis*
using *assms powr-divide*[*of* $1 + cmod\ z\ 1 - cmod\ z\ r$]
using ‹$0 < 1 + cmod\ z$› ‹$(1 + cmod\ z)$ *powr* $r \neq 0$›
unfolding *otimes′-k-def tanh-real-altdef artanh-def*
by (*simp add*: *powr-minus-divide*)
qed

lemma *cmod-otimes′-k*:
assumes $cmod\ z < 1$
shows $cmod\ (otimes'\text{-}k\ r\ z) < 1$
by (*smt assms divide-less-eq-1-pos divide-minus-left otimes′-k-def norm-of-real powr-gt-zero zero-less-norm-iff*)

definition *otimes′* :: *real* ⇒ *complex* ⇒ *complex* **where**
$otimes'\ r\ z = (if\ z = 0\ then\ 0\ else\ cor\ (otimes'\text{-}k\ r\ z) * (z / cmod\ z))$

lemma *cmod-otimes′*:
assumes $cmod\ z < 1$
shows $cmod\ (otimes'\ r\ z) = abs\ (otimes'\text{-}k\ r\ z)$
proof (*cases* $z = 0$)
case *True*
thus *?thesis*
by (*simp add*: *otimes′-def otimes′-k-def*)
next
case *False*
hence $cmod\ (cor\ (otimes'\text{-}k\ r\ z)) = abs\ (otimes'\text{-}k\ r\ z)$
by *simp*
then show *?thesis*
using *False*
unfolding *otimes′-def*
by (*simp add*: *norm-divide norm-mult*)
qed

lift-definition *otimes* :: *real* ⇒ *PoincareDisc* ⇒ *PoincareDisc* (**infixl** ⊗ *105*) **is** *otimes′*
using *cmod-otimes′ cmod-otimes′-k* **by** *auto*

lemma *otimes-distrib-lemma′*:
fixes *ax bx ay by* :: *real*
assumes $ax + bx \neq 0$ $ay + by \neq 0$
shows $(ax * ay - bx * by) / (ax * ay + bx * by) =$
$((ax - bx)/(ax + bx) + (ay - by)/(ay + by)) /$
$(1 + ((ax - bx)/(ax + bx))*((ay - by)/(ay + by)))$ (**is** *?lhs* = *?rhs*)
proof−
have $(ax - bx)/(ax + bx) + (ay - by)/(ay + by) = ((ax - bx)*(ay + by) +$

```
(ay − by)∗(ax + bx)) / ((ax + bx)∗(ay + by))
    by (simp add: ‹ax + bx ≠ 0› ‹ay + by ≠ 0› add-frac-eq)
  hence 1: (ax − bx)/(ax + bx) + (ay − by)/(ay + by) = 2 ∗ (ax ∗ ay − bx ∗
by) / ((ax + bx)∗(ay + by))
    by (simp add: field-simps)

  have 1 + ((ax − bx)/(ax + bx))∗((ay − by)/(ay + by)) =
      (((ax + bx)∗(ay + by)) + (ax − bx)∗(ay − by))/((ax + bx)∗(ay + by))
    by (simp add: ‹ax + bx ≠ 0› ‹ay + by ≠ 0› add-divide-distrib)
  hence 2: 1 + ((ax − bx)/(ax + bx))∗((ay − by)/(ay + by)) = 2 ∗ (ax ∗ ay +
bx ∗ by) / ((ax + bx)∗(ay + by))
    by (simp add: field-simps)

  have ?rhs = 2 ∗ (ax ∗ ay − bx ∗ by) / ((ax + bx) ∗ (ay + by)) /
           (2 ∗ (ax ∗ ay + bx ∗ by) / ((ax + bx) ∗ (ay + by)))
    by (subst 1, subst 2, simp)
  also have ... = (2 ∗ (ax ∗ ay − bx ∗ by) ∗ ((ax + bx) ∗ (ay + by))) /
              (2 ∗ ((ax + bx) ∗ (ay + by)) ∗ (ax ∗ ay + bx ∗ by))
    by auto
  also have ... = ((2 ∗ ((ax + bx) ∗ (ay + by))) ∗ (ax ∗ ay − bx ∗ by)) /
              ((2 ∗ ((ax + bx) ∗ (ay + by))) ∗ (ax ∗ ay + bx ∗ by))
    by (simp add: field-simps)
  also have ... = (ax ∗ ay − bx ∗ by) / (ax ∗ ay + bx ∗ by)
    using ‹ax + bx ≠ 0› ‹ay + by ≠ 0› by auto
  finally
  show ?thesis
    by simp
qed

lemma otimes-distrib-lemma:
  assumes cmod a < 1
  shows otimes'-k (r1 + r2) a = oplus-m' (otimes'-k r1 a) (otimes'-k r2 a)
  unfolding otimes'-k-def oplus-m'-def
  unfolding powr-add
  apply (subst otimes-distrib-lemma')
  apply (smt powr-gt-zero powr-non-neg)
  apply (smt powr-gt-zero powr-non-neg)
  apply simp
  done

lemma otimes-oplus-m-distrib:
  shows (r1 + r2) ⊗ a = r1 ⊗ a ⊕m r2 ⊗ a
proof transfer
  fix r1 r2 a
  assume cmod a < 1
  show otimes' (r1 + r2) a = oplus-m' (otimes' r1 a) (otimes' r2 a)
  proof (cases a = 0)
    case True
    then show ?thesis
```

```
      by (simp add: otimes'-def oplus-m'-def)
  next
    case False
    let ?p = 1 + cmod a and ?m = 1 − cmod a
    have cor (otimes'-k (r1 + r2) a) ∗ a / cor (cmod a) =
          oplus-m' (otimes'-k r1 a) (otimes'-k r2 a) ∗ a / cor (cmod a)
      by (simp add: ‹cmod a < 1› otimes-distrib-lemma)
    moreover
    have cor (otimes'-k r1 a) ∗ cnj a ∗ (cor (otimes'-k r2 a) ∗ a) / (cor (cmod a)
∗ cor (cmod a)) =
          cor (otimes'-k r1 a) ∗ cor (otimes'-k r2 a)
       by (smt False complex-mod-cnj complex-mod-mult-cnj complex-norm-square
mult.commute nonzero-mult-div-cancel-left norm-mult of-real-mult times-divide-times-eq
zero-less-norm-iff)
    ultimately
     show ?thesis
      using False
      unfolding otimes'-def oplus-m'-def
      by (smt complex-cnj-complex-of-real complex-cnj-divide complex-cnj-mult dis-
trib-right times-divide-eq-left times-divide-eq-right times-divide-times-eq)
  qed
qed

lemma otimes-assoc:
  shows (r1 ∗ r2) ⊗ a = r1 ⊗ (r2 ⊗ a)
proof transfer
  fix r1 r2 a
  assume cmod a < 1
  show otimes' (r1 ∗ r2) a = otimes' r1 (otimes' r2 a)
  proof (cases a = 0)
    case True
    then show ?thesis
      by (simp add: otimes'-def)
  next
    case False
    show ?thesis
    proof (cases r2 = 0)
      case True
      thus ?thesis
        by (simp add: ‹cmod a < 1› otimes'-def otimes'-k-tanh)
    next
      case False
      let ?a2 = otimes' r2 a
      let ?k2 = otimes'-k r2 a
      have cmod ?a2 = abs ?k2
        using ‹cmod a < 1› cmod-otimes'
        by blast
      hence cmod ?a2 < 1
        using ‹cmod a < 1› cmod-otimes'-k
```

```
      by auto
    have (1 + cmod a) / (1 − cmod a) > 1
      using ‹a ≠ 0›
      by (simp add: ‹cmod a < 1›)
    hence artanh (cmod a) > 0
      by (simp add: artanh-def)
    hence ?k2 ≠ 0
      using ‹cmod a < 1› ‹a ≠ 0› otimes'-k-tanh[of a r2] ‹r2 ≠ 0›
      by auto
    hence ?a2 ≠ 0
      using ‹a ≠ 0›
      unfolding otimes'-def
      by simp
    have sgn ?k2 = sgn r2
      using otimes'-k-tanh[OF ‹cmod a < 1›, of r2]
    by (smt ‹0 < artanh (cmod a)› ‹cmod ?a2 = |?k2|› ‹?a2 ≠ 0› mult-nonneg-nonneg
mult-nonpos-nonneg sgn-neg sgn-pos tanh-0 tanh-real-neg-iff zero-less-norm-iff)
    have otimes' r1 (otimes' r2 a) =
          cor (otimes'-k r1 (cor ?k2 ∗ a / cor (cmod a))) ∗
          (cor ?k2 ∗ a) / (cor (cmod a) ∗ abs ?k2)
      using False ‹?a2 ≠ 0›
      using ‹cmod ?a2 = |?k2|›
      unfolding otimes'-def
      by auto
    also have ... = cor (tanh (r1 ∗ |r2 ∗ artanh (cmod a)|)) ∗
            (cor ?k2 ∗ a) / (cor (cmod a) ∗ abs ?k2)
      using cmod-otimes'[of a r2] ‹cmod a < 1› ‹a ≠ 0›
      unfolding otimes'-def
      using ‹cmod ?a2 < 1› ‹cmod ?a2 = |?k2|› otimes'-k-tanh
      using ‹cmod a < 1› otimes'-k-tanh[of a r2]
      by (simp add: artanh-abs-tanh)
    also have ... = cor (tanh (r1 ∗ |r2| ∗ artanh (cmod a))) ∗
            (cor ?k2 ∗ a) / (cor (cmod a) ∗ abs ?k2)
      using ‹artanh (cmod a) > 0›
    by (smt ab-semigroup-mult-class.mult-ac(1) mult-minus-left mult-nonneg-nonneg)
    also have ... = cor (tanh (r1 ∗ |r2| ∗ artanh (cmod a))) ∗ sgn ?k2 ∗ (a / cor
(cmod a))
      by (simp add: mult.commute real-sgn-eq)
    also have ... = cor (tanh (r1 ∗ |r2| ∗ artanh (cmod a))) ∗ sgn r2 ∗ (a / cor
(cmod a))
      using ‹sgn ?k2 = sgn r2›
      by simp
    also have ... = cor (tanh (r1 ∗ r2 ∗ artanh (cmod a))) ∗ (a / cor (cmod a))
      by (cases r2 ≥ 0) auto
    finally show ?thesis
      by (simp add: ‹cmod a < 1› otimes'-def otimes'-k-tanh)
  qed
 qed
qed
```

```
lemma otimes-scale-prop:
  fixes r :: real
  assumes r ≠ 0
  shows to-complex (|r| ⊗ a) / ⟪r ⊗ a⟫ = to-complex a / ⟪a⟫
proof−
  let ?f = λ r a. tanh (r ∗ artanh (cmod (to-complex a)))

  have ∗: to-complex (|r| ⊗ a) = ?f |r| a ∗ (to-complex a / ⟪a⟫)
    using Mobius-gyrocarrier'.gyronorm-def to-complex otimes'-def otimes'-k-tanh
otimes.rep-eq
    by force
  then have ⟪r ⊗ a⟫ = cmod (?f r a ∗ (to-complex a / ⟪a⟫))
   by (metis (no-types, lifting) Mobius-gyrocarrier'.gyronorm-def to-complex cmod-otimes'
otimes'-def otimes'-k-tanh otimes.rep-eq mem-Collect-eq norm-mult norm-of-real)
  then have ⟪r ⊗ a⟫ = |?f r a / ⟪a⟫| ∗ cmod (to-complex a)
   by (metis (no-types, opaque-lifting) norm-mult norm-of-real of-real-divide times-divide-eq-left
times-divide-eq-right)

  have ?f |r| a = tanh(|r| ∗ |artanh (cmod (to-complex a))|)
    by (smt (verit) to-complex artanh-nonneg mem-Collect-eq norm-ge-zero)
  then have ?f |r| a = |?f r a|
    by (metis abs-mult tanh-real-abs)

  have |?f r a / ⟪a⟫| = ?f |r| a / ⟪a⟫
    by (metis Mobius-gyrocarrier'.gyronorm-def to-complex abs-divide abs-le-self-iff
abs-mult-pos abs-norm-cancel artanh-nonneg dual-order.refl mem-Collect-eq  tanh-real-abs)
  then have ∗∗:?f |r| a / ⟪a⟫ = |?f r a / ⟪a⟫|
    by simp

  show ?thesis
  proof (cases to-complex a = 0)
    case True
    then show ?thesis
      using assms
      by (simp add: otimes'-def otimes.rep-eq)
  next
    case False
    then have |?f r a / ⟪a⟫| ≠0
      using assms
     by (metis artanh-0 Mobius-gyrocarrier'.gyronorm-def to-complex abs-norm-cancel
artanh-not-0 artanh-tanh-real divide-eq-0-iff linorder-not-less mem-Collect-eq mult-eq-0-iff
norm-eq-zero not-less-iff-gr-or-eq zero-less-abs-iff)
    then show ?thesis
      using ∗ ∗∗ Mobius-gyrocarrier'.gyronorm-def
            ‹⟪r ⊗ a⟫ = |?f r a / ⟪a⟫| ∗ cmod (to-complex a)›
      by fastforce
  qed
qed
```

lemma *gamma-factor-eq1-lemma1*:
shows *cmod(1 + cnj a * b)***cmod(1 + cnj a * b) − cmod(a+b)***cmod(a+b) =
(1 − cmod a * cmod a) * (1 − cmod b * cmod b)*

proof−
have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) = (1+(cnj a)***b) * cnj(1+(cnj a)***b)*
by (*metis complex-norm-square power2-eq-square*)
then have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) = (1+(cnj a)***b)***(1+(cnj(cnj a))** (cnj b))*
using *complex-cnj-add complex-cnj-mult complex-cnj-one* **by** *presburger*
then have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b)= 1+a***(cnj b)+(cnj a)***b + (cnj a)***b***a***(cnj b)*
by (*simp add*: *field-simps*)
moreover
have *cmod(a+b)***cmod(a+b) = (a+b)***cnj(a+b)*
by (*metis complex-norm-square power2-eq-square*)
then have *cmod(a+b)***cmod(a+b) = a***(cnj a) + a***(cnj b) + b***(cnj a) + b***(cnj b)*
by (*simp add*: *field-simps*)
then have *cmod(a+b)***cmod(a+b) = cmod(a)***cmod(a) + a***(cnj b) + b***(cnj a) + cmod(b)***cmod(b)*
by (*metis complex-norm-square power2-eq-square*)
ultimately have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) − cmod(a+b)***cmod(a+b) =
(1+a***(cnj b)+(cnj a)***b + (cnj a)***b***a***(cnj b))−(cmod(a)***cmod(a) + a***(cnj b) + b***(cnj a) + cmod(b)***cmod(b))*
by *auto*
then have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) − cmod(a+b)***cmod(a+b) =
(1+(cnj a)***a***(b***(cnj b)) − cmod(a)***cmod(a) − cmod(b)***cmod(b))*
by *fastforce*
then have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) − cmod(a+b)***cmod(a+b) =
(1+(cmod(a)***cmod(a))***(b***(cnj b))−cmod(a)***cmod(a) − cmod(b)***cmod(b))*
by (*metis* (*mono-tags*, *opaque-lifting*) *complex-norm-square mult.assoc mult.left-commute power2-eq-square*)
then have *cmod(1+(cnj a)***b)***cmod(1+(cnj a)***b) − cmod(a+b)***cmod(a+b) =
(1+(cmod(a)***cmod(a))***(cmod(b)***cmod(b))−cmod(a)***cmod(a) − cmod(b)***cmod(b))*
by (*smt* (*verit*) *Re-complex-of-real cmod-power2 complex-In-mult-cnj-zero complex-mod-cnj complex-mod-mult-cnj diff-add-cancel cnj-cmod norm-mult norm-zero of-real-1 plus-complex.sel*(*1*) *times-complex.sel*(*1*))
moreover
have *(1−cmod(a)***cmod(a))***(1−cmod(b)***cmod(b)) = 1+(cmod(a)***cmod(a))***(cmod(b)***cmod(b))−cmod(a)*
by (*simp add*: *field-simps*)
ultimately
show *?thesis*
by *presburger*
qed

lemma *gamma-factor-eq1-lemma2*:

```
  fixes x y::real
  assumes y > 0
  shows 1 / sqrt(1 − (x∗x)/(y∗y)) = abs y / sqrt(y∗y − x∗x)
proof−
  have 1 − ((x∗x)/(y∗y)) = (y∗y−x∗x) / (y∗y)
    using assms
    by (metis diff-divide-distrib div-0 divide-less-cancel divide-self no-zero-divisors)
  then have sqrt (1 − (x∗x)/(y∗y)) = sqrt(y∗y−x∗x)/sqrt(y∗y)
    using real-sqrt-divide by presburger
  then have sqrt(1 − (x∗x)/(y∗y)) = sqrt(y∗y−x∗x)/abs(y)
    using real-sqrt-abs2 by presburger
  then show ?thesis
    by auto
qed

lemma gamma-factor-norm-oplus-m:
  shows γ («a ⊕m b») =
        γ (to-complex a) ∗
        γ (to-complex b) ∗
        cmod (1 + cnj (to-complex a) ∗ (to-complex b))
proof−
  let ?a = to-complex a and ?b = to-complex b
  have norm («a ⊕m b») < 1
    using Mobius-gyrocarrier′.gyronorm-def to-complex abs-square-less-1
    by fastforce
  then have ∗: γ («a ⊕m b») =
              1 / sqrt(1 − cmod (?a+?b) / cmod (1 + cnj ?a ∗?b) ∗ cmod (?a+?b)
/ cmod (1 + cnj ?a ∗?b))
   using Mobius-gyrocarrier′.gyronorm-def gamma-factor-def oplus-m′-def oplus-m.rep-eq
norm-divide norm-eq-zero norm-le-zero-iff norm-of-real real-norm-def
    by (smt (verit, del-insts) power2-eq-square times-divide-eq-right)
  also have . . . =
          cmod(1 + cnj ?a ∗ ?b) /
          sqrt(cmod (1 + cnj ?a ∗ ?b) ∗ cmod (1 + cnj ?a ∗ ?b) −
               cmod (?a+?b) ∗ cmod (?a+?b))
  proof−
    let ?iz1 = cmod (?a+?b) ∗ cmod (?a+?b)
    let ?iz2 = cmod (1 + cnj ?a ∗ ?b) ∗ cmod (1 + cnj ?a ∗ ?b)
    have ?iz1 ≥ 0
      by force
    moreover
    have ?iz2 > 0
      using den-not-zero to-complex
      by auto
    ultimately show ?thesis
      using zero-less-mult-iff
     by (smt (verit, best) divide-divide-eq-left gamma-factor-eq1-lemma2 norm-not-less-zero
times-divide-eq-left)
  qed
```

also have ... = *cmod*(*1* + *cnj* *?a* ∗ *?b*) / *sqrt*((*1* − *cmod* *?a* ∗ *cmod* *?a*) ∗ (*1* − *cmod* *?b* ∗ *cmod* *?b*))
using *gamma-factor-eq1-lemma1*
by *presburger*
also have ... =
γ (*to-complex* *a*) ∗
γ (*to-complex* *b*) ∗
cmod (*1* + *cnj* (*to-complex* *a*) ∗ (*to-complex* *b*))
proof−
have *cmod* (*to-complex* *a*) < *1* *cmod* (*to-complex* *b*) < *1*
using *to-complex*
by *auto*
then show *?thesis*
unfolding *gamma-factor-def*
by (*simp add*: *power2-eq-square real-sqrt-mult*)
qed
finally show *?thesis*
.
qed

lemma *gamma-factor-norm-oplus-m′*:
shows γ_p (*of-complex* (*cor* (⟪*a* $\oplus_m$ *b*⟫))) =
γ_p (*a*) ∗
γ_p (*b*) ∗
cmod (*1* + *cnj* (*to-complex* *a*) ∗ (*to-complex* *b*))
proof−
have *norm* ((*cor* (⟪*a* $\oplus_m$ *b*⟫))) <*1*
using *norm-lt-one norm-p.rep-eq* **by** *auto*
moreover have γ_p (*of-complex* (*cor* (⟪*a* $\oplus_m$ *b*⟫))) = γ (⟪*a* $\oplus_m$ *b*⟫)
by (*metis Mobius-gyrocarrier′.gyronorm-def Mobius-gyrocarrier′.norm-in-norms Mobius-gyrocarrier-norms-embed′.gyronorm-of-real′ Mobius-gyrocarrier-norms-embed′.of-real′-def gamma-factor-def gammma-factor-p.rep-eq o-apply real-norm-def*)
ultimately show *?thesis*
by (*simp add*: *gamma-factor-norm-oplus-m gammma-factor-p.rep-eq*)
qed

lemma *gamma-factor-oplus-m-triangle-lemma*:
fixes *x y* ::*real*
assumes $x \geq 0$ $x < 1$ $y \geq 0$ $y < 1$
shows *1* / *sqrt* (*1* − ((*x*+*y*)∗(*x*+*y*))/((*1*+*x*∗*y*)∗(*1*+*x*∗*y*))) =
(*1*+*x*∗*y*) / (*sqrt* (*1*−*x*∗*x*) ∗ *sqrt* (*1*−*y*∗*y*))
proof−
have *1* − ((*x*+*y*)∗(*x*+*y*))/((*1*+*x*∗*y*)∗(*1*+*x*∗*y*)) = ((*1*+*x*∗*y*)∗(*1*+*x*∗*y*) − (*x*+*y*)∗(*x*+*y*)) / ((*1*+*x*∗*y*)∗(*1*+*x*∗*y*))
by (*smt* (*verit*, *ccfv-threshold*) *add-divide-distrib assms div-self mult-eq-0-iff mult-nonneg-nonneg*)
then have *1* − ((*x*+*y*)∗(*x*+*y*))/((*1*+*x*∗*y*)∗(*1*+*x*∗*y*)) = ((*1*−*x*∗*x*)∗(*1*−*y*∗*y*)) /

```
((1+x∗y)∗(1+x∗y))
    by (simp add: field-simps)
  then have sqrt(1 − ((x+y)∗(x+y))/((1+x∗y)∗(1+x∗y))) =
           (sqrt(1−x∗x)∗sqrt(1−y∗y)) / (sqrt((1+x∗y)∗(1+x∗y)))
    using assms real-sqrt-divide real-sqrt-mult
    by presburger
  then show ?thesis
    using assms
    by simp
qed

lemma gamma-factor-oplus-m-triangle:
   shows γ («a ⊕m b») ≤ γ (to-complex ((of-complex («a»)) ⊕m (of-complex
(«b»))))
proof−
  have γ (to-complex ((of-complex («a»)) ⊕m (of-complex («b»)))) =
        γ («a») ∗ γ («b») ∗ (1 + «a» ∗ «b»)
  proof−
    let ?expr1 = ((«a») + («b»)) / (1 + («a»)∗(«b»))
    let ?expr2 = to-complex (of-complex («a») ⊕m of-complex («b»))
    have ∗: ?expr1 = ?expr2
     using Mobius-gyrocarrier'.gyronorm-def Mobius-gyrocarrier'.to-carrier to-complex
oplus-m'-def oplus-m.rep-eq
      by auto

    have ∗∗: norm («a») < 1 norm («b») < 1
      using to-complex abs-square-less-1 norm-p.rep-eq
      by fastforce+
    then have ∗∗∗: γ («a») = 1 / sqrt(1 − («a») ∗ («a»))
                 γ («b») = 1 / sqrt(1 − («b») ∗ («b»))
      unfolding gamma-factor-def
      by (auto simp add: power2-eq-square)

    have γ ?expr1 = 1 / sqrt(1 − ((cmod (?expr1)) ∗ cmod(?expr1)))
      using ∗ ∗∗
      unfolding gamma-factor-def power2-eq-square
      by (metis norm-lt-one norm-of-real norm-p.rep-eq real-norm-def)
    moreover
    have cmod ?expr1 = ?expr1
     by (smt (verit, ccfv-threshold) Mobius-gyrocarrier'.gyronorm-def mult-less-0-iff
norm-divide norm-not-less-zero norm-of-real of-real-1 of-real-add of-real-divide of-real-mult)
    ultimately
    have γ ?expr1 = 1 / sqrt (1 − (Re ?expr1 ∗ Re ?expr1))
      by (metis Re-complex-of-real)
   then have γ ?expr1 = (1 + «a»∗«b») / (sqrt (1−«a»∗«a») ∗ sqrt (1−«b»∗«b»))
     using Mobius-gyrocarrier'.gyronorm-def to-complex gamma-factor-oplus-m-triangle-lemma
      using ‹cmod ?expr1 = ?expr1›
      by force
     then have γ (cor ?expr1) = (1 + «a»∗«b») / (sqrt (1−«a»∗«a») ∗ sqrt
```

```
(1−《b》∗《b》))
       unfolding gamma-factor-def
       by (metis norm-of-real real-norm-def)
    then show ?thesis
      using ‹?expr1 = ?expr2›[symmetric]
      using ∗∗∗
      by simp
  qed

  moreover

  have γ (《a》) ∗ γ (《b》) ∗ (1 + 《a》∗《b》) ≥
        γ (to-complex a) ∗ γ (to-complex b) ∗ cmod (1 + cnj (to-complex a) ∗
(to-complex b))
  proof−
    have ∗: γ (《a》) = γ (to-complex a)
            γ (《b》) = γ (to-complex b)
       by (auto simp add: Mobius-gyrocarrier'.gyronorm-def gamma-factor-def)

     have cmod (1 + cnj (to-complex a) ∗ (to-complex b)) ≤
           cmod 1 + cmod (cnj (to-complex a) ∗ (to-complex b))
       using norm-triangle-ineq
       by blast
     also have ... = 1 + cmod (to-complex a) ∗ cmod (to-complex b)
       by (simp add: norm-mult)
     also have ... = 1 + 《a》∗《b》
       using Mobius-gyrocarrier'.gyronorm-def
       by force
     finally show ?thesis
       using ∗[symmetric]
       using Mobius-gyrocarrier'.gyronorm-def gamma-factor-positive norm-lt-one
       by (simp add: gamma-factor-positive)
   qed

   ultimately show ?thesis
     using gamma-factor-norm-oplus-m
     by presburger
qed

lemma mobius-triangle:
  shows 《a ⊕m b》 ≤ 《of-complex (《a》) ⊕m of-complex (《b》)》
proof (cases to-complex a = − to-complex b)
  case True
  then show ?thesis
   by (simp add: Mobius-gyrocarrier'.gyronorm-def oplus-m'-def oplus-m.rep-eq)
next
  case False
  let ?e1 = (《a ⊕m b》)
  let ?e2 = cmod (to-complex (of-complex (《a》) ⊕m of-complex (《b》)))
```

have *?e1* $>$ *0*
by (*smt* (*verit*, *best*) *Mobius-gyrocarrier′.gyronorm-def* ‹*to-complex a* $\neq$ − *to-complex b*› *ab-left-minus add-right-cancel divide-eq-0-iff gamma-factor-norm-oplus-m gamma-factor-positive oplus-m′-def oplus-m.rep-eq norm-eq-zero norm-le-zero-iff of-real-0 zero-less-mult-iff*)
moreover
have *?e2* $>$ *0*
by (*smt* (*verit*, *best*) *calculation gamma-factor-def gamma-factor-increasing gamma-factor-oplus-m-triangle norm-lt-one norm-zero zero-less-norm-iff*)
moreover
have *?e1* $<$ *1* *?e2* $<$ *1*
using *Mobius-gyrocarrier′.gyronorm-def to-complex*
by *auto*
ultimately
show *?thesis*
using *gamma-factor-increase-reverse*[*of ?e1 ?e2*]
by (*smt* (*verit*, *del-insts*) *gamma-factor-def gamma-factor-increasing gamma-factor-oplus-m-triangle norm-p.rep-eq real-norm-def*)
qed

lemma *mobius-triangle′*:
shows $\langle\!\langle a \oplus_m b \rangle\!\rangle \leq$ *Re* (*to-complex* (*of-complex* ($\langle\!\langle a \rangle\!\rangle$) $\oplus_m$ *of-complex* ($\langle\!\langle b \rangle\!\rangle$)))
proof−
have $\langle\!\langle$*of-complex* ($\langle\!\langle a \rangle\!\rangle$) $\oplus_m$ *of-complex* ($\langle\!\langle b \rangle\!\rangle$)$\rangle\!\rangle$ = *Re* (*to-complex* (*of-complex* ($\langle\!\langle a \rangle\!\rangle$) $\oplus_m$ *of-complex* ($\langle\!\langle b \rangle\!\rangle$))) (**is** $\langle\!\langle$*?x*$\rangle\!\rangle$ = *?y*)
proof (*rule gyronorm-Re*)
show *Re* (*to-complex ?x*) $\geq$ *0*
by (*rule oplusM-pos-reals*, *simp-all add*: *norm-geq-zero norm-lt-one*)
next
show *Im* (*to-complex ?x*) = *0*
by (*rule oplusM-reals*, *simp-all add*: *norm-geq-zero norm-lt-one*)
qed
then show *?thesis*
using *mobius-triangle*[*of a b*]
by *simp*
qed

lemma *mobius-gyroauto-norm*:
shows $\langle\!\langle gyr_m\ a\ b\ v \rangle\!\rangle = \langle\!\langle v \rangle\!\rangle$
using *Mobius-gyrocarrier.norm-gyr gyr-PoincareDisc-def*
by *auto*

lemma *otimes-homogenity*:
shows $\langle\!\langle r \otimes a \rangle\!\rangle$ = *cmod* (*to-complex* ($|r|$ $\otimes$ *of-complex* ($\langle\!\langle a \rangle\!\rangle$)))
proof (*cases a* = 0_m)
case *True*
then show *?thesis*
using *Mobius-gyrocarrier′.gyronorm-def otimes′-def otimes.rep-eq ozero-m′-def ozero-m.rep-eq ozero-m-def*

```
    by force
next
  case False
  have ⟪r ⊗ a⟫ = |tanh (r * artanh (⟪a⟫))|
   using Mobius-gyrocarrier'.gyronorm-def to-complex cmod-otimes' otimes'-k-tanh
otimes.rep-eq
    by force
  moreover
  have to-complex (|r| ⊗ of-complex (⟪a⟫)) = tanh (|r| * artanh (⟪a⟫))
  proof−
    have to-complex (|r| ⊗ of-complex (⟪a⟫)) =
          otimes'-k |r| (⟪a⟫) * ((⟪a⟫) / cmod (⟪a⟫))
      using otimes-def otimes'-def
     using Mobius-gyrocarrier'.gyronorm-def Mobius-gyrocarrier'.to-carrier to-complex
otimes.rep-eq
       by (smt (verit, del-insts) False mem-Collect-eq norm-eq-zero norm-geq-zero
norm-of-real of-real-divide of-real-mult to-complex-0-iff)

    moreover
    have cmod (cmod (to-complex a)) = cmod (to-complex a)
      by simp
    then have (⟪a⟫) / cmod (⟪a⟫) = 1
      using ‹a ≠ 0_m›
        by (metis Mobius-gyrocarrier'.gyronorm-def Mobius-gyrocarrier'.of-carrier
div-self norm-eq-zero ozero-m'-def ozero-m.rep-eq)
    ultimately
     have to-complex (|r| ⊗ ( of-complex (cor(⟪a⟫)))) = cor (otimes'-k |r| (cor
(⟪a⟫)))
      by auto
    then show ?thesis
      using Mobius-gyrocarrier'.gyronorm-def to-complex otimes'-k-tanh
      by auto
  qed
  moreover
  have |tanh(r * artanh (cmod (to-complex a))) / ⟪a⟫| =
         tanh (|r| * artanh (cmod (to-complex a))) / ⟪a⟫
    by (metis Mobius-gyrocarrier'.gyronorm-def to-complex abs-divide abs-le-self-iff
abs-mult-pos abs-norm-cancel artanh-nonneg dual-order.refl mem-Collect-eq tanh-real-abs)
  ultimately
  show ?thesis
    by (smt (verit, best) norm-of-real real-compex-cmod tanh-real-abs)
qed

lemma otimes-homogenity':
  shows ⟪r ⊗ a⟫ = Re (to-complex (|r| ⊗ of-complex (⟪a⟫)))
proof−
  have *: cor (⟪a⟫) ∈ {z. cmod z < 1}
    by (simp add: norm-geq-zero norm-lt-one)
  have **: cmod (otimes' |r| (cor (⟪a⟫))) < 1
```

using *cmod-otimes′ cmod-otimes′-k norm-lt-one norm-p.rep-eq*
by *force*

have *Re* (*to-complex* (|*r*| ⊗ *of-complex* (⟪*a*⟫))) ≥ *0*
unfolding *otimes-def*
using *of-complex-inverse Mobius-gyrocarrier′.to-carrier*[*OF* ∗] ∗∗ ∗
by (*simp add*: *artanh-nonneg norm-geq-zero otimes′-def otimes′-k-tanh*)

moreover

have *Im* (*to-complex* (|*r*| ⊗ *of-complex* (⟪*a*⟫))) = *0*
unfolding *otimes-def*
using *of-complex-inverse Mobius-gyrocarrier′.to-carrier*[*OF* ∗] ∗∗ ∗
by (*simp add*: *otimes′-def*)

ultimately

have *Re* (*to-complex* (|*r*| ⊗ *of-complex* (⟪*a*⟫))) = *cmod* (*to-complex* (|*r*| ⊗ *of-complex* (⟪*a*⟫)))
using *cmod-eq-Re*
by *force*

then show *?thesis*
using *otimes-homogenity*[*of r a*]
by *simp*
qed

lemma *gyr-m-gyrospace*:
shows gyr_m (*r1* ⊗ *v*) (*r2* ⊗ *v*) = *id*
proof−
have *gyr-m′* (*to-complex* (*r1* ⊗ *v*)) (*to-complex* (*r2* ⊗ *v*)) = *id*
proof−
let *?v* = *to-complex v*
let *?e1* = *?v* ∗ *tanh* (*r1* ∗ *artanh* (*cmod ?v*)) /*cmod*(*to-complex v*)
let *?e2* = *?v* ∗ *tanh* (*r2* ∗ *artanh* (*cmod ?v*)) /*cmod*(*to-complex v*)

have *to-complex* (*r1* ⊗ *v*) = *?e1*
to-complex (*r2* ⊗ *v*) = *?e2*
using *to-complex otimes′-def otimes′-k-tanh otimes.rep-eq*
by *auto*

moreover

have *cnj ?e1* = *cnj ?v* ∗ *cnj* (*tanh* (*r1* ∗ *artanh* (*cmod ?v*))) / *cmod ?v*
cnj ?e2 = *cnj ?v* ∗ *cnj* (*tanh* (*r2* ∗ *artanh* (*cmod ?v*))) / *cmod ?v*
by *auto*

moreover

have $(1 + ?e1 * (cnj\ ?e2)) / (1 + ?e2 * (cnj\ ?e1)) = 1$
proof−
have $1 + ?e1 * (cnj\ ?e2) = 1 + ?e2 * (cnj\ ?e1)$
by *simp*
moreover
have $1 + ?e2 * (cnj\ ?e1) \neq 0$
using ‹*to-complex* $(r1 \otimes v) = ?e1$› ‹*to-complex* $(r2 \otimes v) = ?e2$›
by (*metis to-complex div-by-0 divide-eq-1-iff mem-Collect-eq cmod-mix-cnj norm-zero*)
ultimately
show *?thesis*
by *simp*
qed

ultimately show *?thesis*
using *gyr*$_m$*-def gyr-m*$'$*-def*
by (*metis eq-id-iff mult.commute mult-1*)
qed

then show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *add-0-left add-0-right complex-cnj-zero div-by-1 eq-id-iff gyr*$_m$*-def m-left-id oplus-m*$'$*-def oplus-m-def ozero-m*$'$*-def ozero-m.rep-eq map-fun-apply mult-zero-left*)
qed

lemma *gyr-m-gyrospace2*:
shows $gyr_m\ u\ v\ (r \otimes a) = r \otimes (gyr_m\ u\ v\ a)$
proof−
let $?u = to\text{-}complex\ u$ **and** $?v = to\text{-}complex\ v$ **and** $?a = to\text{-}complex\ a$
let $?e1 = gyr_m\ u\ v\ a$
let $?e2 = cmod\ (to\text{-}complex\ ?e1)$

have $?e1 = of\text{-}complex\ ((1 + ?u * cnj\ ?v) / (1 + cnj\ ?u * ?v) * ?a)$
by (*metis Mobius-gyrocarrier*$'$*.of-carrier gyr-m*$'$*-def gyr*$_m$*.rep-eq*)
then have $?e2 = cmod\ ((1 + ?u * cnj\ ?v) / (1 + cnj\ ?u * ?v)) * cmod\ ?a$
by (*metis gyr-m*$'$*-def gyr*$_m$*.rep-eq norm-mult*)
then have $?e2 = cmod\ ?a$
using *Mobius-gyrocarrier*$'$*.gyronorm-def mobius-gyroauto-norm*
by *presburger*
then have $r \otimes ?e1 = of\text{-}complex\ (((1{+}cmod\ ?a)\ powr\ r - (1{-}cmod\ ?a)\ powr\ r) / ((1{+}cmod\ ?a)\ powr\ r + (1{-}cmod\ ?a)\ powr\ r) * to\text{-}complex\ ?e1 / ?e2)$
using *otimes-def*
by (*metis* (*no-types*, *lifting*) *Mobius-gyrocarrier*$'$*.of-carrier otimes*$'$*-def otimes*$'$*-k-def otimes.rep-eq mult-eq-0-iff times-divide-eq-right*)
then have $r \otimes ?e1 = of\text{-}complex\ (to\text{-}complex\ (r \otimes a) * ((1 + ?u * cnj\ ?v) / (1 + cnj\ ?u * ?v)))$
using *otimes-def*

```
    using ‹?e2 = cmod ?a›
   by (smt (verit, ccfv-threshold) Mobius-gyrocarrier'.of-carrier ab-semigroup-mult-class.mult-ac(1)
gyr-m'-def gyr_m.rep-eq otimes'-def otimes'-k-def otimes.rep-eq mult.commute mult-eq-0-iff
times-divide-eq-right)
  then show ?thesis
    by (metis Mobius-gyrocarrier'.of-carrier gyr-m'-def gyr_m.rep-eq mult.commute)
qed


lemma reals':
  shows cor ` {x. abs x < 1} = {z. cmod z < 1 ∧ Im z = 0}
  by (auto, simp add: complex-eq-iff norm-complex-def)

lemma zero-times-m [simp]:
  shows 0 ⊗ x = 0_m
  by transfer (simp add: otimes'-def otimes'-k-tanh ozero-m'-def)

interpretation Mobius-gyrocarrier-norms-embed: gyrocarrier-norms-embed to-complex
otimes
proof
  fix a b
 assume a ∈ gyrocarrier-norms-embed'.reals to-complex b ∈ gyrocarrier-norms-embed'.reals
to-complex
  then obtain x y where a = of-complex (cor x) b = of-complex (cor y) abs x <
1 abs y < 1
    by auto
  then show a ⊕ b ∈ gyrocarrier-norms-embed'.reals to-complex
   by (metis (mono-tags, lifting) Mobius-gyrocarrier'.of-carrier Mobius-gyrocarrier'.to-carrier
Mobius-gyrocarrier-norms-embed'-reals gyroplus-PoincareDisc-def image-eqI mem-Collect-eq
oplusM-reals reals' to-complex)
next
  fix a
  assume a ∈ gyrocarrier-norms-embed'.reals to-complex
  then obtain x where a = of-complex (cor x) abs x < 1
    by auto
  then have ⊖ a = of-complex (cor (−x)) abs (−x) < 1
    unfolding gyroinv-PoincareDisc-def
    unfolding ominus-m-def
   by (metis Mobius-gyrocarrier'.of-carrier Mobius-gyrocarrier'.to-carrier map-fun-apply
mem-Collect-eq norm-of-real of-real-minus ominus-m'-def ominus-m.rep-eq to-complex-inverse,
simp)
  then show ⊖ a ∈ gyrocarrier-norms-embed'.reals to-complex
    by auto
next
  fix r a
  assume a ∈ gyrocarrier-norms-embed'.reals to-complex
  then obtain x where a = of-complex (cor x) abs x < 1
    by auto
  then have a = PoincareDisc.of-complex (cor x)  abs x < 1
```

using *Mobius-gyrocarrier-norms-embed′.of-real′-def Mobius-gyrocarrier-norms-embed′-of-real′*
by *auto*
have *Im* (*to-complex* (*r* ⊗ *a*)) = *0*
by (*simp add*: ‹|*x*| < *1*› ‹*a* = *MobiusGyroVectorSpace.of-complex* (*cor x*)›
otimes′-def otimes.rep-eq)
moreover
have *abs* (*Re* (*to-complex* (*r* ⊗ *a*))) < *1*
by (*metis Mobius-gyrocarrier′.gyronorm-def calculation cmod-eq-Re norm-lt-one*)
ultimately
show *r* ⊗ *a* ∈ *gyrocarrier-norms-embed′.reals to-complex*
unfolding *Mobius-gyrocarrier-norms-embed′-of-real′*
by (*metis* (*mono-tags*, *lifting*) *Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′-reals image-eqI mem-Collect-eq reals′ to-complex*)
qed

interpretation *Mobius-pre-gyrovector-space*: *pre-gyrovector-space to-complex otimes*
proof
fix *a* :: *PoincareDisc*
show *1* ⊗ *a* = *a*
by *transfer* (*auto simp add*: *otimes′-def otimes′-k-def*)
next
fix *r1 r2 a*
show (*r1* + *r2*) ⊗ *a* = *r1* ⊗ *a* ⊕ *r2* ⊗ *a*
using *gyroplus-PoincareDisc-def otimes-oplus-m-distrib* **by** *auto*
next
fix *r1 r2 a*
show (*r1* ∗ *r2*) ⊗ *a* = *r1* ⊗ (*r2* ⊗ *a*)
by (*simp add*: *otimes-assoc*)
next
fix *r* :: *real* **and** *a*
assume *r* ≠ *0*
then show *to-complex* (*abs r* ⊗ *a*) $/_R$ *gyrocarrier′.gyronorm to-complex* (*r* ⊗ *a*)
=
to-complex a $/_R$ *gyrocarrier′.gyronorm to-complex a*
using *otimes-scale-prop*[*of r a*]
by (*metis Mobius-gyrocarrier′.gyrocarrier′-axioms divide-inverse gyrocarrier′.gyronorm-def mult.commute norm-p.rep-eq of-real-inverse scaleR-conv-of-real*)
next
fix *u v r a*
have gyr_m *u v* (*r* ⊗ *a*) = *r* ⊗ gyr_m *u v a*
using *gyr-m-gyrospace2*
by *auto*
then show *gyr u v* (*r* ⊗ *a*) = *r* ⊗ *gyr u v a*
using *gyr-PoincareDisc-def* **by** *auto*
next
fix *r1 r2 v*
have gyr_m (*r1* ⊗ *v*) (*r2* ⊗ *v*) = *id*
using *gyr-m-gyrospace*
by *simp*

then show *gyr* (*r1* ⊗ *v*) (*r2* ⊗ *v*) = *id*
by (*simp add*: *gyr-PoincareDisc-def*)
qed

interpretation *Mobius-gyrovector-space*: *gyrovector-space-norms-embed otimes to-complex*

proof
fix *r a*
show *gyrocarrier′.gyronorm to-complex* (*r* ⊗ *a*) =
Mobius-gyrocarrier-norms-embed.otimesR |*r*| (*gyrocarrier′.gyronorm to-complex*
a)
using *otimes-homogenity′*[*of r a*]
by (*smt* (*verit*, *best*) *Im-eq-0 Mobius-gyrocarrier′.gyrocarrier′-axioms Mobius-gyrocarrier′.gyronorm-def*
Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′.of-real′-def Mobius-gyrocarrier-norms-embed′.
Mobius-gyrocarrier-norms-embed.otimesR-def abs-Re-le-cmod add.right-neutral comp-apply
complex-eq gyrocarrier′.gyronorm-def mult-zero-right of-real-0 otimes-homogenity)
next
fix *a b*
show *gyrocarrier′.gyronorm to-complex* (*a* ⊕ *b*)
≤ *Mobius-gyrocarrier-norms-embed′.oplusR* (*gyrocarrier′.gyronorm to-complex*
a) (*gyrocarrier′.gyronorm to-complex b*)
proof−
have *Re* (*to-complex* (*of-complex* (*cmod* (*to-complex a*)) $\oplus_m$ *of-complex* (*cmod*
(*to-complex b*)))) =
Mobius-gyrocarrier-norms-embed′.to-real′ (*Mobius-gyrocarrier-norms-embed′.of-real′*
(*cmod* (*to-complex a*)) ⊕ *Mobius-gyrocarrier-norms-embed′.of-real′* (*cmod* (*to-complex*
b)))
by (*smt* (*verit*, *ccfv-threshold*) *Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′-of-real′*
Mobius-gyrocarrier-norms-embed′-to-real′ complex-mod-minus-le-complex-mod gy-
roplus-PoincareDisc-def image-eqI mem-Collect-eq of-complex-inverse oplusM-reals
reals′ to-complex)
then show *?thesis*
using *mobius-triangle′*[*of a b*]
by (*simp add*: *Mobius-gyrocarrier′.gyrocarrier′-axioms Mobius-gyrocarrier′.gyronorm-def*
Mobius-gyrocarrier-norms-embed′.gyrocarrier-norms-embed′-axioms gyrocarrier′.gyronorm-def
gyrocarrier-norms-embed′.oplusR-def gyroplus-PoincareDisc-def)
qed
qed

lemma *norm-scale-tanh*:
shows ⟪*r* ⊗ *z*⟫ = |*tanh* (*r* ∗ *artanh* (⟪*z*⟫))|
proof *transfer*
fix *r z*
assume *cmod z* < *1*
have *cmod* ((*otimes′-k r z*) ∗ *z* / *cor* (*cmod z*)) = *cmod* (*otimes′-k r z*)
by (*smt* (*verit*) *artanh-0 div-by-0 mult-cancel-right1 nonzero-eq-divide-eq norm-divide*

norm-not-less-zero norm-of-real of-real-0 otimes'-k-tanh tanh-0)
  then show *cmod* (*otimes' r z*) = |*tanh* (*r* ∗ *artanh* (*cmod z*))|
    unfolding *otimes'-def*
    using ‹*cmod z* < *1*› *otimes'-k-tanh*
    by *auto*
qed

lemma *ominus-m-scale*:
  shows *k* ⊗ ($\ominus_m$ *u*) = $\ominus_m$ (*k* ⊗ *u*)
  using *Mobius-pre-gyrovector-space.scale-minus' gyroinv-PoincareDisc-def*
  by *auto*

lemma *otimes-2-oplus-m*: *2* ⊗ *u* = *u* $\oplus_m$ *u*
  using *Mobius-pre-gyrovector-space.times2 gyroplus-PoincareDisc-def*
  by *simp*

definition *half'* :: *complex* ⇒ *complex* **where**
  half' v = (γ *v* / (*1* + γ *v*)) $*_R$ *v*

lift-definition *half* :: *PoincareDisc* ⇒ *PoincareDisc* **is** *half'*
  unfolding *half'-def*
proof−
  fix *v*
  assume *cmod v* < *1*
  let *?k* = γ *v* / (*1* + γ *v*)
  have *abs ?k* < *1*
    using ‹*cmod v* < *1*› *gamma-factor-positive* **by** *fastforce*
  then show *cmod* (*?k* $*_R$ *v*) < *1*
    using ‹*cmod v* < *1*›
    by (*metis mult-closed-for-unit-disc norm-of-real scaleR-conv-of-real*)
qed

lemma *otimes-2-half*:
  shows *2* ⊗ (*half v*) = *v*
proof−
  have *2* ⊗ (*half v*) = *half v* $\oplus_m$ *half v*
    using *otimes-2-oplus-m*
    by *simp*
  also have . . . = *v*
  proof *transfer*
    fix *v* :: *complex*
    assume *assms*: *cmod v* < *1*
    have ∗: γ *v* ≠ *0 1* + γ *v* ≠ *0*
      using *assms gamma-factor-positive*
      by *fastforce*+
    let *?k* = γ *v* / (*1* + γ *v*)
    have *1* + *cnj* (*?k* ∗ *v*) ∗ (*?k* ∗ *v*) = *1* + *?k*^*2* ∗ (*cmod v*)^2
      by (*simp add*: *cnj-cmod mult.commute power2-eq-square*)
    also have . . . = *1* + (γ *v*)^2 / (*1* + γ *v*)^2 ∗ (*1* − *1* / (γ *v*)^2)

using *norm-square-gamma-factor*[*OF assms*]
by (*simp add*: *power-divide*)
also have ... $= 1 + ((\gamma\ v)^2 * ((\gamma\ v)^2 - 1)) / ((\gamma\ v)^2 * (1 + \gamma\ v)^2)$
using $*$
by (*simp add*: *field-simps*)
also have ... $= 1 + ((\gamma\ v)^2 - 1) / (1 + \gamma\ v)^2$
using $*$
by *simp*
also have ... $= 1 + ((\gamma\ v - 1) * (\gamma\ v + 1)) / ((\gamma\ v + 1) * (\gamma\ v + 1))$
by (*simp add*: *power2-eq-square field-simps*)
also have ... $= 1 + (\gamma\ v - 1) / (\gamma\ v + 1)$
using $*$
by *simp*
also have ... $= 2 * ?k$
using $*$
by (*simp add*: *field-simps*)
finally show *oplus-m′* (*half′ v*) (*half′ v*) = *v*
unfolding *oplus-m′-def half′-def*
using $*$ ‹$1 + cnj\ (?k * v) * (?k * v) = 1 + ?k^2 * (cmod\ v)^2$›
by (*smt* (*verit*) *mult-eq-0-iff nonzero-mult-div-cancel-left of-real-eq-0-iff power2-eq-square scaleR-conv-of-real scaleR-left-distrib*)
qed
finally show *?thesis*
.
qed

lemma *half*:
shows *half v* = $(1/2) \otimes v$
by (*metis Mobius-pre-gyrovector-space.scale-assoc mult-2 real-scaleR-def scaleR-half-double otimes-2-half*)

lemma *half′*:
assumes *cmod u* < *1*
shows *otimes′* (*1*/*2*) *u* = *half′ u*
using *assms half half.rep-eq*[*of of-complex u*] *otimes.rep-eq*
by *simp*

lemma *half-gamma′*:
shows *to-complex* $((1\ /\ 2) \otimes u) =$
$(\gamma\ (to\text{-}complex\ u)) / (1 + \gamma\ (to\text{-}complex\ u)) * to\text{-}complex\ u$
using *half half.rep-eq half′-def*
by (*simp add*: *scaleR-conv-of-real*)

definition *double′* :: *complex* ⇒ *complex* **where**
double′ v $= (2 * (\gamma\ v)^2 / (2 * (\gamma\ v)^2 - 1)) *_R v$

lemma *double′-cmod*:
assumes *cmod v* < *1*
shows $2 * (\gamma\ v)^2 / (2 * (\gamma\ v)^2 - 1) = 2 / (1 + (cmod\ v)^2)$ (**is** *?lhs* = *?rhs*)

proof−
  have ∗∗: $1 - (cmod\ v)^2 > 0$
    using *assms*
    using *real-sqrt-lt-1-iff* **by** *fastforce*

  have $?lhs = 2 * (1 / (1 - (cmod\ v)^2)) / (2 * (1 / (1 - (cmod\ v)^2)) - 1)$
    using *gamma-factor-square-norm*[*OF assms*]
    by *simp*
  also have $\ldots = 2 / (1 + (cmod\ v)^2)$
  proof−
    have $2 * (1 / (1 - (cmod\ v)^2)) = 2 / (1 - (cmod\ v)^2)$
      by *simp*
    moreover
    have $2 * (1 / (1 - (cmod\ v)^2)) - 1 = 2 / (1 - (cmod\ v)^2) - (1 - (cmod\ v)^2) / (1 - (cmod\ v)^2)$
      using ∗∗
      by *simp*
    then have $2 * (1 / (1 - (cmod\ v)^2)) - 1 = (1 + (cmod\ v)^2) / (1 - (cmod\ v)^2)$
      using ∗∗
      by (*simp add*: *field-simps*)
    ultimately
    show *?thesis*
      using ∗∗
        by (*smt* (*verit*, *del-insts*) *divide-divide-eq-left nonzero-mult-div-cancel-left power2-eq-square times-divide-eq-right*)
  qed
  finally show *?thesis*
    .
qed

lemma *cmod-double′*:
  assumes $cmod\ v < 1$
  shows $cmod\ (double'\ v) = 2 * cmod\ v / (1 + (cmod\ v)^2)$
proof−
  have $cmod\ (double'\ v) =$
      $abs(2 * (\gamma\ v)^2 / (2 * (\gamma\ v)^2 - 1)) * cmod\ v$
    unfolding *double′-def*
    by *simp*
  also have $\ldots = abs\ (2 / (1 + (cmod\ v)^2)) * cmod\ v$
    using *assms double′-cmod*
    by *presburger*
  also have $\ldots = 2 * cmod\ v / (1 + (cmod\ v)^2)$
  proof−
    have $2 / (1 + (cmod\ v)^2) > 0$
      by (*metis half-gt-zero-iff power-one sum-power2-gt-zero-iff zero-less-divide-iff zero-neq-one*)
    then show *?thesis*
      by *simp*

qed
finally show *?thesis*
.
qed

lift-definition *double* :: *PoincareDisc* ⇒ *PoincareDisc* **is** *double′*
proof−
fix *v*
assume ∗: *cmod v* < *1*

have *cmod* (*double′ v*) = *2* ∗ *cmod v* / (*1* + (*cmod v*)^2)
using ∗ *cmod-double′*
by *simp*
also have . . . < *1*
proof−
have (*1* − *cmod v*)^2 > *0*
using ∗
by *simp*
then have *1* − *2*∗ *cmod v* + (*cmod v*)^2 > *0*
by (*simp add*: *field-simps power2-eq-square*)
then have *2*∗*cmod v* < *1* + (*cmod v*)^2
by *simp*
moreover
have *1* + (*cmod v*)^2 > *0*
by (*smt* (*verit*) *not-sum-power2-lt-zero*)
ultimately
show *?thesis*
using *divide-less-eq-1* **by** *blast*
qed
finally
show *cmod* (*double′ v*) < *1*
by *simp*
qed

lemma *double′-otimes′-2*:
assumes *cmod v* < *1*
shows *double′ v* = *otimes′ 2 v*
proof−
have *v* ∗ *2* / (*1* + *cor* (*cmod v*) ∗ *cor* (*cmod v*)) =
v ∗ *4* / (*2* + *2* ∗ (*cor* (*cmod v*) ∗ *cor* (*cmod v*)))

by (*metis* (*no-types*, *lifting*) *distrib-left-numeral mult-2 nonzero-mult-divide-mult-cancel-left numeral-Bit0 one-add-one times-divide-eq-right zero-neq-numeral*)

then show *?thesis*
using *assms*
unfolding *double′-def otimes′-def otimes′-k-def double′-cmod*[*OF assms*] *scaleR-conv-of-real*
by (*auto simp add*: *field-simps power2-eq-square*)
qed

lemma *double*:
  shows *double u* = *2* ⊗ *u*
  by *transfer* (*simp add*: *double′-otimes′-2*)

end
theory *Einstein*
  imports *Complex-Main GyroGroup GyroVectorSpace GyroVectorSpaceIsomorphism GammaFactor HOL.Real-Vector-Spaces*
  MobiusGyroGroup MobiusGyroVectorSpace HOL.Transcendental
begin

Einstein zero

definition *ozero-e′* :: *complex* **where**
  ozero-e′ = *0*

lift-definition *ozero-e* :: *PoincareDisc* (0_e) **is** *ozero-e′*
  unfolding *ozero-e′-def*
  by *simp*

lemma *ozero-e-ozero-m*:
  shows $0_e = 0_m$
  using *ozero-e′-def ozero-e-def ozero-m′-def ozero-m-def*
  by *auto*

Einstein addition

definition *oplus-e′* :: *complex* ⇒ *complex* ⇒ *complex* **where**
  oplus-e′ u v = (*1* / (*1* + *inner u v*)) $*_R$ (*u* + (*1* / γ *u*) $*_R$ *v* + ((γ *u* / (*1* + γ *u*)) ∗ (*inner u v*)) $*_R$ *u*)

lemma *noroplus-m′-e*:
  assumes *norm u* < *1 norm v* <*1*
  shows *norm* (*oplus-e′ u v*)^*2* =
      1 / (*1* + *inner u v*)^*2* ∗ (*norm*(*u*+*v*)^*2* − ((*norm u*)^*2* ∗(*norm v*)^*2* − (*inner u v*)^*2*))
proof−
  let *?uv* = *inner u v*
  let *?gu* = γ *u* / (*1* + γ *u*)

  have *1*: *norm* (*oplus-e′ u v*)^*2* =
      norm (*1* / (*1* + *?uv*))^*2* ∗ *norm* ((*u* + ((*1* / γ *u*) $*_R$ *v*) + (*?gu* ∗ *?uv*) $*_R$ *u*))^*2*
    by (*metis oplus-e′-def norm-scaleR power-mult-distrib real-norm-def*)

  have *2*: *norm* (*1* / (*1* + *?uv*))^*2* = *1* / (*1* + *?uv*)^*2*
    by (*simp add*: *power-one-over*)

  have *norm*((*u* + ((*1* / γ *u*) $*_R$ *v*) + (*?gu* ∗ *?uv*) $*_R$ *u*))^*2* =

```
      inner (u + (1 / γ u) *R v + (?gu * ?uv) *R u)
           (u + (1 / γ u) *R v + (?gu * ?uv) *R u)
    by (simp add: dot-square-norm)
  also have ... =
      (norm u)^2 +
      (norm ((1 / γ u) *R v))^2 +
      (norm ((?gu * ?uv) *R u))^2 +
      2 * inner u ((1 / γ u) *R v) +
      2 * inner u ((?gu * ?uv) *R u) +
      2 * inner ((?gu * ?uv) *R u) ((1 / γ u) *R v) (is ?lhs = ?a + ?b + ?c +
?d + ?e + ?f)
    by (smt (verit) inner-commute inner-right-distrib power2-norm-eq-inner)
  also have ... = (norm u)^2 +
               1 / (γ u)^2 * (norm v)^2 +
               ?gu^2 * (inner u v)^2 * (norm u)^2 +
               2 / γ u * (inner u v) +
               2 * ?gu * ?uv * (inner u u) +
               2 * ?gu * ?uv * (1 / γ u) * (inner u v)
  proof−
    have ?b = 1 / (γ u)^2 * (norm v)^2
      by (simp add: power-divide)
    moreover
    have ?c = ?gu^2 * (inner u v)^2 * (norm u)^2
      by (simp add: power2-eq-square)
    moreover
    have ?d = 2 / γ u * (inner u v)
      using inner-scaleR-right
      by auto
    moreover
    have ?e = 2 * ?gu * ?uv * (inner u u)
      using inner-scaleR-right
      by auto
    moreover
    have ?f = 2 * ?gu * ?uv * (1 / γ u) * (inner u v)
      by force
    ultimately
    show ?thesis
      by presburger
  qed
  also have ... = 2 * inner u v + (inner u v)^2 + (norm u)^2 + (1 − (norm
u)^2) * (norm v)^2 (is ?a + ?b + ?c + ?d + ?e + ?f = ?rhs)
  proof−
    have ?a + ?b = (norm u)^2 + (1 − (norm u)^2) * (norm v)^2
      using assms norm-square-gamma-factor
      by force

    moreover have ?d + ?e = 2 * inner u v (is ?lhs = ?rhs)
    proof−
      have ?e = 2 * (γ u * (norm u)^2 / (1 + γ u)) * inner u v
```

```
      by (simp add: dot-square-norm)
    moreover
    have 1 / γ u + γ u ∗ (norm u)^2 / (1 + γ u) = 1
      using assms(1) gamma-expression-eq-one-1
      by blast
    moreover
    have ?d + 2 ∗ (γ u ∗ (norm u)^2 / (1 + γ u)) ∗ inner u v = 2 ∗ inner u v
∗ (1 / γ u + γ u ∗ (norm u)^2 / (1 + γ u))
      by (simp add: distrib-left)
    ultimately
    show ?thesis
      by (metis mult.right-neutral)
  qed

  moreover

  have ?c + ?f = (inner u v)^2
  proof−
    have ?c + ?f = ?gu^2 ∗ (norm u)^2 ∗ (inner u v)^2 + 2 ∗ (1 / γ u) ∗ ?gu
∗ (inner u v)^2
      by (simp add: mult.commute mult.left-commute power2-eq-square)
    then have ?c + ?f = ((γ u / (1 + γ u))^2 ∗ (norm u)^2 + 2 ∗ (1 / γ u)
∗ (γ u / (1 + γ u))) ∗ (inner u v)^2
      by (simp add: ring-class.ring-distribs(2))
    moreover
    have (γ u / (1 + γ u))^2 ∗ (norm u)^2 + 2 ∗ (1 / γ u) ∗ (γ u / (1 + γ
u)) = 1
    proof −
      have ∀ (x::real) y n. (x / y) ^ n = x ^ n / y ^ n
        by (simp add: power-divide)
      then show ?thesis
        using gamma-expression-eq-one-2[OF assms(1)]
        by fastforce
    qed
    ultimately
    show ?thesis
      by simp
  qed

  ultimately
  show ?thesis
    by auto
qed
also have ... = ((cmod (u + v))² − ((cmod u)² ∗ (cmod v)² − ?uv²))
  unfolding dot-square-norm[symmetric]
  by (simp add: inner-commute inner-right-distrib field-simps)
finally
have 3: norm ((u + ((1 / γ u) ∗R v) + (?gu ∗ ?uv) ∗R u))^2 =
        norm(u+v)^2 − ((norm u)^2 ∗(norm v)^2 − ?uv^2)
```

```
    by simp

  show ?thesis
    using 1 2 3
    by simp
qed

lemma gamma-oplus-e':
  assumes norm u < 1 norm v < 1
  shows 1 / sqrt(1 − norm (oplus-e' u v)^2) = γ u ∗ γ v ∗ (1 + inner u v)
proof−
  let ?uv = inner u v

  have abs: abs (1 + ?uv) = 1 + ?uv
    using abs-inner-lt-1 assms by fastforce

  have 1 − norm (oplus-e' u v)^2 =
        1 − 1 / (1 + ?uv)^2 ∗ (norm(u+v)^2 − ((norm u)^2 ∗(norm v)^2 −
?uv^2))
    using assms noroplus-m'-e
    by presburger
  also have . . . = ((1 + ?uv)^2 − (norm(u+v)^2 − ((norm u)^2 ∗(norm v)^2
− ?uv^2))) /
              (1 + ?uv)^2
  proof−
    have ?uv ≠ −1
      using abs-inner-lt-1[OF assms]
      by auto
    then have (1 + ?uv)^2 ≠ 0
      by auto
    then show ?thesis
      by (simp add: diff-divide-distrib)
  qed
  also have . . . = (1 − (norm u)^2 − (norm v)^2 + (norm u)^2 ∗ (norm v)^2)
/ (1 + ?uv)^2
  proof−
    have (1 + ?uv)^2 = 1 + 2∗?uv + ?uv^2
      by (simp add: power2-eq-square field-simps)
    moreover
    have norm(u+v)^2 − ((norm u)^2 ∗(norm v)^2 − ?uv^2) =
        (norm u)^2 + 2∗?uv + (norm v)^2 − (norm u)^2∗(norm v)^2 + ?uv^2
      by (smt (z3) dot-norm field-sum-of-halves)
    ultimately
    show ?thesis
      by auto
  qed
  finally have 1 / sqrt (1 − norm (oplus-e' u v)^2) =
            1 / sqrt((1 − (norm u)^2 − (norm v)^2 + (norm u)^2∗(norm v)^2)
/ (1 + ?uv)^2)
```

```
    by simp
  then have 1: 1 / sqrt (1 − norm (oplus-e′ u v)^2) =
           (1 + ?uv) / sqrt (1 − (norm u)^2 − (norm v)^2 + (norm u)^2*(norm
v)^2)
    using abs
    by (simp add: real-sqrt-divide)

  have γ u = 1 / sqrt(1 − (norm u)^2) γ v = 1 / sqrt(1 − (norm v)^2)
    using assms
    by (metis gamma-factor-def)+
  then have γ u * γ v = (1 / sqrt (1 − (norm u)^2)) * (1 / sqrt (1 − (norm
v)^2))
    by simp
  also have ... = 1 / sqrt ((1 − (norm u)^2) * (1 − (norm v)^2))
    by (simp add: real-sqrt-mult)
  finally have 2: γ u * γ v = 1 / sqrt ((1 − (norm u)^2 − (norm v)^2 + (norm
u)^2*(norm v)^2))
    by (simp add: field-simps power2-eq-square)

  show ?thesis
    using 1 2
    by (metis (no-types, lifting) mult-cancel-right1 times-divide-eq-left)
qed

lemma gamma-oplus-e′-not-zero:
  assumes norm u < 1 norm v < 1
  shows 1 / sqrt(1 − norm(oplus-e′ u v)^2) ≠ 0
  using assms
  using gamma-oplus-e′ gamma-factor-def gamma-factor-nonzero noroplus-m′-e
  by (smt (verit, del-insts) divide-eq-0-iff mult-eq-0-iff zero-eq-power2)

lemma oplus-e′-in-unit-disc:
  assumes norm u < 1 norm v < 1
  shows norm (oplus-e′ u v) < 1
proof−
  let ?uv = inner u v
  have 1 + ?uv > 0
    using abs-inner-lt-1[OF assms]
    by fastforce
  then have γ u * γ v * (1 + inner u v) > 0
    using gamma-factor-positive[OF assms(1)]
          gamma-factor-positive[OF assms(2)]
    by fastforce
  then have 0 < sqrt (1 − (cmod (oplus-e′ u v))²)
    using gamma-oplus-e′[OF assms] gamma-oplus-e′-not-zero[OF assms]
    by (metis zero-less-divide-1-iff)
  then have (norm (oplus-e′ u v))^2 < 1
    using real-sqrt-gt-0-iff
    by simp
```

then show *?thesis*
using *real-less-rsqrt* **by** *force*
qed

lemma *gamma-factor-oplus-e′*:
assumes *norm u < 1 norm v < 1*
shows γ (*oplus-e′ u v*) = (γ *u*) ∗ (γ *v*) ∗ (*1* + *inner u v*)
proof−
have γ (*oplus-e′ u v*) = *1* / *sqrt*(*1* − *norm* (*oplus-e′ u v*)^*2*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *oplus-e′-in-unit-disc gamma-factor-def*)
then show *?thesis*
using *assms*
using *gamma-oplus-e′* **by** *force*
qed

lift-definition *oplus-e* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc* (**infixl** $\oplus_e$ *100*) **is** *oplus-e′*
by (*rule oplus-e′-in-unit-disc*)

definition *ominus-e′* :: *complex* ⇒ *complex* **where**
ominus-e′ v = − *v*

lemma *ominus-e′-in-unit-disc*:
assumes *norm z < 1*
shows *norm* (*ominus-e′ z*) < *1*
using *assms*
unfolding *ominus-e′-def*
by *simp*

lift-definition *ominus-e* :: *PoincareDisc* ⇒ *PoincareDisc* ($\ominus_e$) **is** *ominus-e′*
using *ominus-e′-in-unit-disc* **by** *blast*

lemma *ominus-e-ominus-m*:
shows $\ominus_e$ *a* = $\ominus_m$ *a*
by (*simp add*: *ominus-e′-def ominus-e-def ominus-m′-def ominus-m-def*)

lemma *ominus-e-scale*:
shows *k* ⊗ ($\ominus_e$ *u*) = $\ominus_e$ (*k* ⊗ *u*)
using *ominus-e-ominus-m ominus-m-scale* **by** *auto*

lemma *gamma-factor-p-positive*:
shows γ_p *a* > *0*
by *transfer* (*simp add*: *gamma-factor-positive*)

lemma *gamma-factor-p-oplus-e*:

shows γ_p $(u \oplus_e v) = \gamma_p\ u * \gamma_p\ v * (1 + u \cdot v)$
using *gamma-factor-oplus-e′*
by *transfer blast*

abbreviation γ_2 :: *complex* ⇒ *real* **where**
$\gamma_2\ u \equiv \gamma\ u\ /\ (1 + \gamma\ u)$

lemma *norm-square-gamma-half-scale*:
assumes *norm u* < *1*
shows $(norm\ (\gamma_2\ u *_R u))^2 = (\gamma\ u - 1)\ /\ (1 + \gamma\ u)$
proof−
have $(norm\ (\gamma_2\ u *_R u))^2 = (\gamma_2\ u)^2 * (norm\ u)^2$
by (*simp add*: *power2-eq-square*)
also have ... $= (\gamma_2\ u)^2 * ((\gamma\ u)^2 - 1)\ /\ (\gamma\ u)^2$
using *assms*
by (*simp add*: *norm-square-gamma-factor′*)
also have ... $= (\gamma\ u)^2\ /\ (1 + \gamma\ u)^2 * ((\gamma\ u)^2 - 1)\ /\ (\gamma\ u)^2$
by (*simp add*: *power-divide*)
also have ... $= ((\gamma\ u)^2 - 1)\ /\ (1 + \gamma\ u)^2$
using *assms gamma-factor-positive*
by *fastforce*
also have ... $= (\gamma\ u - 1) * (\gamma\ u + 1)\ /\ (1 + \gamma\ u)^2$
by (*simp add*: *power2-eq-square square-diff-one-factored*)
also have ... $= (\gamma\ u - 1)\ /\ (1 + \gamma\ u)$
by (*simp add*: *add.commute power2-eq-square*)
finally
show *?thesis*
by *simp*
qed

lemma *norm-half-square-gamma*:
assumes *norm u* < *1*
shows $(norm\ (half'\ u))^2 = (\gamma_2\ u)^2 * (cmod\ u)^2$
unfolding *half′-def*
using *norm-square-gamma-half-scale assms*
by (*smt* (*verit*) *divide-pos-pos gamma-factor-positive norm-scaleR power-mult-distrib*)

lemma *norm-half-square-gamma′*:
assumes *cmod u* < *1*
shows $(norm\ (half'\ u))^2 = (\gamma\ u - 1)\ /\ (1 + \gamma\ u)$
using *assms*
using *half′-def norm-square-gamma-half-scale*
by *auto*

lemma *inner-half-square-gamma*:
assumes *cmod u* < *1 cmod v* < *1*
shows *inner* (*half′ u*) (*half′ v*) $= \gamma_2\ u * \gamma_2\ v *$ *inner u v*
unfolding *half′-def scaleR-conv-of-real*
by (*metis inner-mult-left inner-mult-right mult.assoc*)

```
lemma iso-me-help1:
  assumes norm v < 1
  shows 1 + (γ v − 1) / (1 + γ v) = 2 ∗ γ v / (1 + γ v)
proof−
  have 1 + γ v ≠ 0
    using assms gamma-factor-positive
    by fastforce
  then show ?thesis
    by (smt (verit, del-insts) diff-divide-distrib divide-self)
qed

lemma  iso-me-help2:
  assumes norm v < 1
  shows 1 − (γ v − 1) / (1 + γ v) = 2 / (1 + γ v)
proof−
  have 1 + γ v ≠ 0
    using assms gamma-factor-positive
    by fastforce
  then show ?thesis
    by (smt (verit, del-insts) diff-divide-distrib divide-self)
qed

lemma  iso-me-help3:
  assumes norm v < 1 norm u <1
  shows 1 + ((γ v − 1) / (1 + γ v)) ∗ ((γ u − 1) / (1 + γ u)) =
        2 ∗ (1 + (γ u) ∗ (γ v)) / ((1 + γ v) ∗ (1 + γ u)) (is ?lhs = ?rhs)
proof−
  have ∗: 1 + γ v ≠ 0 1 + γ u ≠ 0
    using assms gamma-factor-positive by fastforce+
  have (1 + γ v) ∗ (1 + γ u) = 1 + (γ v) + (γ u) + (γ u)∗(γ v)
    by (simp add: field-simps)
  moreover
  have (γ v − 1) ∗ (γ u − 1) =  (γ u)∗(γ v) − (γ u) − (γ v) +1
    by (simp add: field-simps)
  moreover
  have ?lhs = ((1 + γ v) ∗ (1 + γ u) + (γ u − 1) ∗ (γ v − 1)) / ((1 + γ v) ∗
(1 + γ u))
    using ∗
    by (simp add: add-divide-distrib)
  ultimately show ?thesis
    by (simp add: mult.commute)
qed

lemma half'-oplus-e':
  fixes u v :: complex
  assumes cmod u < 1 cmod v < 1
  shows half' (oplus-e' u v) =
        γ u ∗ γ v / (γ u ∗ γ v ∗ (1 + inner u v) + 1) ∗ (u + (1 / γ u) ∗ v + (γ
```

u / (1 + γ u)) * *inner u v* * u)
proof−
  have *half′* (*oplus-e′ u v*) =
    γ *u* * γ *v* * (1 + *inner u v*) / (γ *u* * γ *v* * (1 + *inner u v*) + 1) *
    ((1 / (1 + *inner u v*)) * (*u* + (1 / γ *u*)*v + (γ *u* / (1 + γ *u*)) * *inner u v*
* *u*))
    unfolding *half′-def*
    unfolding *gamma-factor-oplus-e′*[*OF assms*] *scaleR-conv-of-real*
    unfolding *oplus-e′-def scaleR-conv-of-real*
    by *simp*
  then show *?thesis*
    using *assms*
   by (*smt* (*verit*, *best*) *ab-semigroup-mult-class.mult-ac*(1) *gamma-oplus-e′ gamma-oplus-e′-not-zero inner-mult-left′ inner-real-def mult.commute mult-eq-0-iff nonzero-mult-divide-mult-cancel-right2 of-real-1 of-real-divide of-real-mult real-inner-1-right times-divide-times-eq*)
qed

lemma *oplus-m′-half′*:
  fixes *u v* :: *complex*
  assumes *cmod u* < 1 *cmod v* < 1
  shows *oplus-m′* (*half′ u*) (*half′ v*) =
    (γ *u* * γ *v* / (γ *u* * γ *v* * (1 + *inner u v*) + 1)) *
    (*u* + (1 / γ *u*) * *v* + (γ *u* / (1 + γ *u*) * *inner u v*) * *u*)
proof−
  have *: γ *u* ≠ 0 γ *v* ≠ 0 1 + γ *u* ≠ 0 1 + γ *v* ≠ 0
    using *assms gamma-factor-positive*
    by *fastforce*+

  let *?den* = (1 + γ *v*) * (1 + γ *u*)
  let *?DEN* = γ *u* * γ *v* * (1 + *inner u v*) + 1
  let *?NOM* = *u* + (1 / γ *u*) * *v* + (γ *u* / (1 + γ *u*) * *inner u v*) * *u*

  have **: *cmod* (*half′ u*) < 1 *cmod* (*half′ v*) < 1
    using *assms*
    by (*metis eq-onp-same-args half.rsp rel-fun-eq-onp-rel*)+
  then have *oplus-m′* (*half′ u*) (*half′ v*) = *oplus-m′-alternative* (*half′ u*) (*half′ v*)
    by (*simp add*: *oplus-m′-alternative*)
  also have . . . = ((2 * γ_2 *v* + 2 * γ_2 *v* * γ_2 *u* * *inner u v*) * γ_2 *u* * *u* + 2 *
γ *v* / *?den* * *v*) /
      (2 * γ *u* * γ *v* * *inner u v* / *?den* + 2 * (1 + γ *u* * γ *v*) / *?den*)
  proof−
    have (1 + 2 * *inner* (*half′ u*) (*half′ v*) + (*norm* (*half′ v*))^2) $*_R$ (*half′ u*) =
      (2 * γ_2 *v* + 2 * γ *v* * γ *u* / *?den* * *inner u v*) * γ_2 *u* * *u*
    proof−
      have *: *half′ u* = (γ *u* / (1 + γ *u*)) * *u*
        by (*simp add*: *half′-def scaleR-conv-of-real*)

      have 1 + 2 * *inner* (*half′ u*) (*half′ v*) + (*cmod* (*half′ v*))^2 =
        1 + 2 * (γ_2 *u* * γ_2 *v* * *inner u v*) + (γ_2 *v*)^2 * (*cmod v*)^2

using *inner-half-square-gamma norm-half-square-gamma assms*
by *simp*
also have ... $= 2 * \gamma\ v\ /\ (1 + \gamma\ v) + 2 * \gamma\ v * \gamma\ u\ /\ ?den * inner\ u\ v$
using *assms norm-half-square-gamma norm-square-gamma-half-scale*[*OF assms*(*2*)] *iso-me-help1*[*OF assms*(*2*)] *half′-def*
by (*smt* (*verit*, *best*) *add-divide-distrib distrib-left inner-commute inner-left-distrib inner-real-def times-divide-times-eq*)
finally
show *?thesis*
using ∗
by (*simp add*: *of-real-def*)
qed
moreover
have $(1 - (norm\ (half'\ u))^2)\ \ *_R\ (half'\ v) =$
$(\ 2 * (\gamma\ v)\ /\ ?den) * v$
proof−
have $(norm\ (half'\ u))^2 = (\gamma\ u - 1)\ /\ (1 + \gamma\ u)$
using *assms*(*1*) *norm-half-square-gamma′* **by** *blast*
moreover have $1 - (\gamma\ u - 1)\ /\ (1 + \gamma\ u) = 2/\ \ (1 + \gamma\ u)$
using *assms*(*1*) *iso-me-help2* **by** *blast*
ultimately show *?thesis*
by (*simp add*: *half′-def mult.commute scaleR-conv-of-real*)
qed

moreover
have $1 + 2 * inner\ (half'\ u)\ (half'\ v) + (cmod\ (half'\ u))^2 * (cmod\ (half'\ v))^2 =$
$2 * \gamma\ u * \gamma\ v * inner\ u\ v\ /\ ?den + 2 * (1 + \gamma\ u * \gamma\ v)\ /\ ?den$
using *assms inner-half-square-gamma iso-me-help3 norm-half-square-gamma′*
by (*simp add*: *field-simps*)
ultimately
show *?thesis*
unfolding *oplus-m′-alternative-def*
by (*simp add*: *mult.commute*)
qed
also have ... $= (2 * \gamma\ v * \gamma\ u * u + 2 * \gamma\ v * \gamma\ u * inner\ u\ v * \gamma_2\ u * u + 2 * \gamma\ v * v)\ /$
$(2 * \gamma\ u * \gamma\ v * inner\ u\ v + (2 + 2 * \gamma\ u * \gamma\ v))$
proof−
have $1\ /\ ?den \neq 0$
using ∗
by *simp*
moreover
have $(2 * \gamma_2\ v + 2 * \gamma_2\ v * \gamma_2\ u * inner\ u\ v) * \gamma_2\ u * u + 2 * \gamma\ v\ /\ ?den * v =$
$(1\ /\ ?den) * (2 * \gamma\ v * \gamma\ u * u + 2 * \gamma\ v * \gamma\ u * inner\ u\ v * \gamma_2\ u * u + 2 * \gamma\ v * v)$
by (*simp add*: *mult.commute ring-class.ring-distribs*(*1*))
moreover

have $2 * \gamma\ u * \gamma\ v * inner\ u\ v\ /\ ?den + 2 * (1 + \gamma\ u * \gamma\ v)\ /\ ?den =$
$(1\ /\ ?den) * (2 * \gamma\ u * \gamma\ v * inner\ u\ v + (2 + 2 * \gamma\ u * \gamma\ v))$
by *argo*
ultimately
show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *divide-divide-eq-left′ division-ring-divide-zero eq-divide-eq inner-commute inner-real-def mult-eq-0-iff mult-eq-0-iff nonzero-mult-divide-mult-cancel-left nonzero-mult-divide-mult-cancel-left numeral-One of-real-1 of-real-1 of-real-divide of-real-inner-1 of-real-mult one-divide-eq-0-iff real-inner-1-right times-divide-times-eq*)
qed
also have $\ldots = 2 * (\gamma\ v * \gamma\ u * u + \gamma\ v * \gamma\ u * inner\ u\ v * \gamma\ u\ /\ (1 + \gamma\ u) * u + \gamma\ v * v)\ /\ (2 * ?DEN)$
by (*simp add*: *field-simps*)
also have $\ldots = (\gamma\ v * \gamma\ u * u + \gamma\ v * \gamma\ u * inner\ u\ v * \gamma\ u\ /\ (1 + \gamma\ u) * u + \gamma\ v * v)\ /\ ?DEN$
by (*metis* (*no-types*, *opaque-lifting*) *nonzero-mult-divide-mult-cancel-left of-real-mult of-real-numeral zero-neq-numeral*)
also have $\ldots = ((\gamma\ v * \gamma\ u) * u + (\gamma\ v * \gamma\ u) * (inner\ u\ v * \gamma\ u\ /\ (1 + \gamma\ u) * u) + (\gamma\ u * \gamma\ v) * (v\ /\ \gamma\ u))\ /\ ?DEN$
using $\langle \gamma\ u \neq 0 \rangle$
by *simp*
also have $\ldots = (\gamma\ v * \gamma\ u) * ?NOM\ /\ ?DEN$
proof−
have $(\gamma\ v * \gamma\ u) * u + (\gamma\ v * \gamma\ u) * (inner\ u\ v * \gamma\ u\ /\ (1 + \gamma\ u) * u) + (\gamma\ u * \gamma\ v) * (v\ /\ \gamma\ u) = (\gamma\ v * \gamma\ u) * ?NOM$
by (*simp add*: *field-simps*)
then show *?thesis*
by *simp*
qed
finally show *?thesis*
by *simp*
qed

lemma *iso-me-oplus*:
shows $(1/2) \otimes (u \oplus_e v) = ((1/2) \otimes u) \oplus_m ((1/2) \otimes v)$
proof *transfer*
fix *u v*
assume $*$: $cmod\ u < 1\ cmod\ v < 1$
have $otimes'\ (1\ /\ 2)\ (oplus\text{-}e'\ u\ v) = half'\ (oplus\text{-}e'\ u\ v)$
using $half'[of\ oplus\text{-}e'\ u\ v]\ *$
unfolding *otimes′-def*
using *oplus-e′-in-unit-disc*
by *blast*
moreover
have $otimes'\ (1\ /\ 2)\ u = half'\ u\ otimes'\ (1\ /\ 2)\ v = half'\ v$
using $half'\ *$
by *auto*
moreover
have $half'\ (oplus\text{-}e'\ u\ v) = oplus\text{-}m'\ (half'\ u)\ (half'\ v)$

using $*$ *half′-oplus-e′*[*OF* $*$] *oplus-m′-half′*[*OF* $*$]
by *simp*
ultimately
show *otimes′* (*1* / *2*) (*oplus-e′ u v*) = *oplus-m′* (*otimes′* (*1* / *2*) *u*) (*otimes′* (*1* / *2*) *v*)
by *simp*
qed

lemma *oplus-e-oplus-m*:
shows $u \oplus_e v = 2 \otimes ((1/2) \otimes u \oplus_m (1/2) \otimes v)$
by (*metis half iso-me-oplus otimes-2-half*)

lemma *iso-two-me-oplus*:
shows $2 \otimes (u \oplus_m v) = (2 \otimes u) \oplus_e (2 \otimes v)$
by (*metis Mobius-pre-gyrovector-space.double-half iso-me-oplus otimes-2-oplus-m*)

lemma *iso-two-me-ominus*:
shows $2 \otimes (\ominus_m u) = \ominus_e (2 \otimes u)$
using *ominus-e-ominus-m ominus-e-scale* **by** *auto*

lemma *iso-two-me-zero*:
shows $2 \otimes 0_m = 0_e$
using *Mobius-pre-gyrovector-space.times-zero gyrozero-PoincareDisc-def ozero-e-ozero-m*
by *fastforce*

lemma *iso-two-me-bij*:
shows *bij* (λ *x*::*PoincareDisc*. $2 \otimes x$)
by (*metis Mobius-pre-gyrovector-space.equation-solving bijI′ half otimes-2-half*)

definition gyr_e::*PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc* ⇒ *PoincareDisc*
where
$gyr_e\ u\ v\ w = \ominus_e (u \oplus_e v) \oplus_e (u \oplus_e (v \oplus_e w))$

typedef *PoincareDiscM* = *UNIV*::*PoincareDisc set*
by *auto*
setup-lifting *type-definition-PoincareDiscM*

lift-definition *zero-M* :: *PoincareDiscM* (0_M) **is** 0_m **.**

lift-definition *ominus-M* :: *PoincareDiscM* ⇒ *PoincareDiscM* ($\ominus_M$) **is** ($\ominus_m$) **.**

lift-definition *oplus-M* :: *PoincareDiscM* ⇒ *PoincareDiscM* ⇒ *PoincareDiscM* (**infixl** $\oplus_M$ *100*) **is** ($\oplus_m$) **.**

lift-definition *gyr-M* :: *PoincareDiscM* ⇒ *PoincareDiscM* ⇒ *PoincareDiscM* ⇒ *PoincareDiscM* **is** gyr_m **.**

lift-definition *to-complex-M* :: *PoincareDiscM* ⇒ *complex* **is** *to-complex* **.**

interpretation *gyrogroupoid-M*: *gyrogroupoid zero-M oplus-M* **.**

instantiation *PoincareDiscM* :: *gyrogroupoid*
begin
definition *gyrozero-PoincareDiscM* **where** *gyrozero-PoincareDiscM* = 0_M
definition *gyroplus-PoincareDiscM* **where** *gyroplus-PoincareDiscM* = *oplus-M*
instance
  ..
end

instantiation *PoincareDiscM* :: *gyrocommutative-gyrogroup*
begin
definition *gyroinv-PoincareDiscM* **where** *gyroinv-PoincareDiscM* = *ominus-M*
definition *gyr-PoincareDiscM* **where** *gyr-PoincareDiscM* = *gyr-M*
instance proof
  fix *a* :: *PoincareDiscM*
  show $0_g \oplus a = a$
    unfolding *gyrozero-PoincareDiscM-def gyroplus-PoincareDiscM-def*
    by *transfer auto*
next
  fix *a* :: *PoincareDiscM*
  show $\ominus\ a \oplus a = 0_g$
   unfolding *gyrozero-PoincareDiscM-def gyroplus-PoincareDiscM-def gyroinv-PoincareDiscM-def*
    by *transfer auto*
next
  fix *a b z* :: *PoincareDiscM*
  show $a \oplus (b \oplus z) = a \oplus b \oplus$ *gyr a b z*
    unfolding *gyroplus-PoincareDiscM-def gyr-PoincareDiscM-def*
    by *transfer* (*simp add*: *gyr-m-left-assoc*)
next
  fix *a b* :: *PoincareDiscM*
  show *gyr a b* = *gyr* $(a \oplus b)$ *b*
    unfolding *gyroplus-PoincareDiscM-def gyr-PoincareDiscM-def*
    using *gyr-m-left-loop*
    by *transfer auto*
next
  fix *a b* :: *PoincareDiscM*
  show *gyroaut* (*gyr a b*)
   unfolding *gyroplus-PoincareDiscM-def gyr-PoincareDiscM-def gyroaut-def bij-def*
inj-def surj-def
    by *transfer* (*metis gyr-m-distrib gyr-m-inv*)
next
  fix *a b* :: *PoincareDiscM*
  show $a \oplus b$ = *gyr a b* $(b \oplus a)$
    unfolding *gyroplus-PoincareDiscM-def gyr-PoincareDiscM-def*
    by *transfer* (*metis gyr-m-commute*)
qed

end

typedef *PoincareDiscE* = *UNIV*::*PoincareDisc set*
  by *auto*
setup-lifting *type-definition-PoincareDiscE*

lift-definition *zero-E* :: *PoincareDiscE* (0_E) **is** 0_e **.**

lift-definition *ominus-E* :: *PoincareDiscE* ⇒ *PoincareDiscE* ($\ominus_E$) **is** ($\ominus_e$) **.**

lift-definition *oplus-E* :: *PoincareDiscE* ⇒ *PoincareDiscE* ⇒ *PoincareDiscE* (**infixl** $\oplus_E$ *100*) **is** ($\oplus_e$) **.**

lift-definition *gyr-E* :: *PoincareDiscE* ⇒ *PoincareDiscE* ⇒ *PoincareDiscE* ⇒ *PoincareDiscE* **is** gyr_e **.**

lift-definition *to-complex-E* :: *PoincareDiscE* ⇒ *complex* **is** *to-complex* **.**

lift-definition φ_{ME} :: *PoincareDiscM* ⇒ *PoincareDiscE* **is** λ *x*::*PoincareDisc*. *2* ⊗ *x* **.**

interpretation *Einstein-gyrogroup-iso*:
  gyrogroup-isomorphism φ_{ME} *zero-E oplus-E ominus-E*
  rewrites
  Einstein-gyrogroup-iso.gyr′ = *gyr-E*
proof−
  show ∗: *gyrogroup-isomorphism* φ_{ME} 0_E ($\oplus_E$) $\ominus_E$
  proof
    show φ_{ME} 0_g = 0_E
      unfolding *gyrozero-PoincareDiscM-def*
      by *transfer* (*simp add*: *iso-two-me-zero*)
  next
    fix *a b*
    show φ_{ME} (*a* ⊕ *b*) = φ_{ME} *a* $\oplus_E$ φ_{ME} *b*
      unfolding *gyroplus-PoincareDiscM-def*
      by *transfer* (*simp add*: *iso-two-me-oplus*)
  next
    fix *a*
    show φ_{ME} (⊖ *a*) = $\ominus_E$ (φ_{ME} *a*)
      unfolding *gyroinv-PoincareDiscM-def*
      by *transfer* (*simp add*: *iso-two-me-ominus*)
  next
    show *bij* φ_{ME}
      unfolding *bij-def*
      by *transfer* (*meson bij-betw-def iso-two-me-bij*)
  qed

  show *gyrogroup-isomorphism.gyr′* ($\oplus_E$) $\ominus_E$ = *gyr-E*
    unfolding *gyrogroup-isomorphism.gyr′-def*[*OF* ∗]

```
    by transfer (force simp add: gyr_e-def)
qed


instantiation PoincareDiscE :: gyrogroupoid
begin
definition gyrozero-PoincareDiscE where gyrozero-PoincareDiscE = 0_E
definition gyroplus-PoincareDiscE where gyroplus-PoincareDiscE = oplus-E
instance
  ..
end

instantiation PoincareDiscE :: gyrocommutative-gyrogroup
begin
definition gyroinv-PoincareDiscE where gyroinv-PoincareDiscE = ominus-E
definition gyr-PoincareDiscE where gyr-PoincareDiscE = gyr-E
instance proof
  fix a :: PoincareDiscE
  show 0_g ⊕ a = a
    unfolding gyrozero-PoincareDiscE-def gyroplus-PoincareDiscE-def
    by simp
next
  fix a :: PoincareDiscE
  show ⊖ a ⊕ a = 0_g
   unfolding gyrozero-PoincareDiscE-def gyroplus-PoincareDiscE-def gyroinv-PoincareDiscE-def
    by simp
next
  fix a b z :: PoincareDiscE
  show a ⊕ (b ⊕ z) = a ⊕ b ⊕ gyr a b z
    unfolding gyroplus-PoincareDiscE-def gyr-PoincareDiscE-def
    by (simp add: Einstein-gyrogroup-iso.gyro-left-assoc)
next
  fix a b :: PoincareDiscE
  show gyr a b = gyr (a ⊕ b) b
    unfolding gyroplus-PoincareDiscE-def gyr-PoincareDiscE-def
    using Einstein-gyrogroup-iso.gyr-left-loop
    by simp
next
  fix a b :: PoincareDiscE
  show gyroaut (gyr a b)
    unfolding gyr-PoincareDiscE-def
   by (metis Einstein-gyrogroup-iso.gyr-gyroaut gyroaut-def gyrogroupoid.gyroaut-def
gyroplus-PoincareDiscE-def)
next
  fix a b :: PoincareDiscE
  show a ⊕ b = gyr a b (b ⊕ a)
    unfolding gyr-PoincareDiscE-def gyroplus-PoincareDiscE-def
    using Einstein-gyrogroup-iso.gyro-commute
    by blast
```

qed

end

lift-definition *scale-M* :: *real* ⇒ *PoincareDiscM* ⇒ *PoincareDiscM* **is** (⊗) **.**

lift-definition *scale-E* :: *real* ⇒ *PoincareDiscE* ⇒ *PoincareDiscE* **is** (⊗) **.**

lemma *gyrocarrier′M*:
  shows *gyrocarrier′ to-complex-M*
proof
  show *inj to-complex-M*
    by *transfer simp*
next
  show *to-complex-M* $0_g = 0$
   by (*simp add*: *gyrozero-PoincareDiscM-def ozero-e-ozero-m ozero-m′-def ozero-m.rep-eq to-complex-M.abs-eq zero-M-def*)
qed

lemma *gyrocarrier-norms-embed′M*:
  shows *gyrocarrier-norms-embed′ to-complex-M*
proof
  show *cor* ‘ *gyrocarrier′.norms to-complex-M* ⊆ *gyrocarrier′.carrier to-complex-M*
    unfolding *gyrocarrier′.norms-def*[*OF gyrocarrier′M*]
    unfolding *gyrocarrier′.gyronorm-def*[*OF gyrocarrier′M*]
    unfolding *gyrocarrier′.carrier-def*[*OF gyrocarrier′M*]
    apply *transfer*
   using *Mobius-gyrocarrier′.carrier-def Mobius-gyrocarrier-norms-embed′.norms-carrier norms*
    by *argo*
next
  show *inj to-complex-M*
    using *gyrocarrier′.inj-to-carrier gyrocarrier′M* **by** *auto*
next
  show *to-complex-M* $0_g = 0$
    by (*simp add*: *gyrocarrier′.to-carrier-zero gyrocarrier′M*)
qed

lemma *of-carrier-M*:
  assumes *cmod z* < *1*
 shows *gyrocarrier′.of-carrier to-complex-M z* = *Abs-PoincareDiscM* (*PoincareDisc.of-complex z*)
  using *assms*
  unfolding *gyrocarrier′.of-carrier-def*[*OF gyrocarrier′M*] *to-complex-M.rep-eq*
 by (*metis* (*mono-tags*, *lifting*) *Rep-PoincareDiscM-inject f-inv-into-f mem-Collect-eq of-complex-inverse rangeI to-complex-M.abs-eq to-complex-M.rep-eq to-complex-inverse*)

global-interpretation *GCM*: *gyrocarrier-norms-embed′ to-complex-M*

rewrites *GCM.norms* = {*x*. *abs x* < *1*} **and**
GCM.reals = *Abs-PoincareDiscM* ' *PoincareDisc.of-complex* ' *cor* ' {*x*. |*x*| < *1*}
defines *of-complex-M* = *gyrocarrier'.of-carrier to-complex-M*
proof−
show ∗: *gyrocarrier-norms-embed' to-complex-M*
using *gyrocarrier-norms-embed'M*
by *simp*

show *norms*: *gyrocarrier'.norms to-complex-M* = {*x*. |*x*| < *1*}
using *to-complex*
unfolding *gyrocarrier'.norms-def*[*OF gyrocarrier'M*] *gyrocarrier'.gyronorm-def*[*OF gyrocarrier'M*]
unfolding *to-complex-M.rep-eq*
by *auto* (*metis* (*no-types*, *lifting*) *abs-eq-iff abs-norm-cancel mem-Collect-eq norm-of-real to-complex-M.abs-eq to-complex-M.rep-eq to-complex-cases*)

show *gyrocarrier-norms-embed'.reals to-complex-M* = *Abs-PoincareDiscM* ' *PoincareDisc.of-complex* ' *cor* ' {*x*. |*x*| < *1*}
using *of-carrier-M*
unfolding *gyrocarrier-norms-embed'.reals-def*[*OF gyrocarrier-norms-embed'M*] *norms*
by (*smt* (*verit*) *Mobius-gyrocarrier-norms-embed'.norms-carrier image-cong image-image mem-Collect-eq subsetD*)
qed

lemma *of-real'-M*:
assumes *abs x* < *1*
shows *GCM.of-real' x* = *Abs-PoincareDiscM* (*PoincareDisc.of-complex* (*cor x*))
using *assms*
by (*simp add*: *GCM.of-real'-def of-carrier-M of-complex-M-def*)

lemma *to-real'-M*:
assumes *z* ∈ *GCM.reals*
shows *GCM.to-real' z* = *Re* (*to-complex-M z*)
using *assms*
unfolding *GCM.reals-def GCM.to-real'-def*
using *GCM.norms-carrier*
by *fastforce*

lemma *gyronorm-M-lt-1* [*simp*]:
shows *abs* (*GCM.gyronorm a*) < *1*
using *GCM.gyronorm-def to-complex to-complex-M.rep-eq* **by** *auto*

lemma *gyrocarrier'-norms-M* [*simp*]:
shows *gyrocarrier'.norms to-complex-M* = *GCM.norms*
by (*simp add*: *GCM.gyrocarrier'-axioms GCM.norms-def gyrocarrier'.norms-def*)

lemma *gyrocarrier-norms-embed'-reals-M* [*simp*]:

shows *gyrocarrier-norms-embed′.reals to-complex-M* = *GCM.reals*
by (*simp add*: *GCM.reals-def gyrocarrier-norms-embed′.reals-def gyrocarrier-norms-embed′M of-complex-M-def*)

lemma *gyrocarrier′E*:
shows *gyrocarrier′ to-complex-E*
proof
show *inj to-complex-E*
by *transfer simp*
next
show *to-complex-E* 0_g = *0*
by (*simp add*: *gyrozero-PoincareDiscE-def ozero-e-ozero-m ozero-m′-def ozero-m.rep-eq to-complex-E.abs-eq zero-E-def*)
qed

lemma *gyrocarrier-norms-embed′E*:
shows *gyrocarrier-norms-embed′ to-complex-E*
proof
show *inj to-complex-E*
by *transfer simp*
next
show *to-complex-E* 0_g = *0*
by (*simp add*: *gyrozero-PoincareDiscE-def ozero-e-ozero-m ozero-m′-def ozero-m.rep-eq to-complex-E.abs-eq zero-E-def*)
next
show *cor* ‘ *gyrocarrier′.norms to-complex-E* ⊆ *gyrocarrier′.carrier to-complex-E*
unfolding *gyrocarrier′.norms-def*[*OF gyrocarrier′E*]
unfolding *gyrocarrier′.gyronorm-def*[*OF gyrocarrier′E*]
unfolding *gyrocarrier′.carrier-def*[*OF gyrocarrier′E*]
apply *transfer*
using *Mobius-gyrocarrier′.carrier-def Mobius-gyrocarrier-norms-embed′.norms-carrier norms*
by *argo*
qed

lemma *of-carrier-E*:
assumes *cmod z* < *1*
shows *gyrocarrier′.of-carrier to-complex-E z* = *Abs-PoincareDiscE* (*PoincareDisc.of-complex z*)
using *assms*
unfolding *gyrocarrier′.of-carrier-def*[*OF gyrocarrier′E*] *to-complex-E.rep-eq*
by (*metis* (*mono-tags*, *lifting*) *Rep-PoincareDiscE-inject f-inv-into-f mem-Collect-eq of-complex-inverse rangeI to-complex-E.abs-eq to-complex-E.rep-eq to-complex-inverse*)

global-interpretation *GCE*: *gyrocarrier-norms-embed′ to-complex-E*
rewrites *GCE.norms* = {*x. abs x* < *1*} **and**
GCE.reals = *Abs-PoincareDiscE* ‘ *PoincareDisc.of-complex* ‘ *cor* ‘ {*x.* |*x*| < *1*}
defines *of-complex-E* = *gyrocarrier′.of-carrier to-complex-E*

proof−
show ∗: *gyrocarrier-norms-embed′ to-complex-E*
using *gyrocarrier-norms-embed′E*
by *simp*

show *norms*: *gyrocarrier′.norms to-complex-E* = {*x*. |*x*| < *1*}
using *to-complex*
unfolding *gyrocarrier′.norms-def*[*OF gyrocarrier′E*] *gyrocarrier′.gyronorm-def*[*OF gyrocarrier′E*]
unfolding *to-complex-E.rep-eq*
by *auto* (*metis* (*no-types*, *lifting*) *abs-eq-iff abs-norm-cancel mem-Collect-eq norm-of-real to-complex-E.abs-eq to-complex-E.rep-eq to-complex-cases*)

show *gyrocarrier-norms-embed′.reals to-complex-E* = *Abs-PoincareDiscE* ‘ *PoincareDisc.of-complex* ‘ *cor* ‘ {*x*. |*x*| < *1*}
using *of-carrier-E*
unfolding *gyrocarrier-norms-embed′.reals-def*[*OF* ∗] *norms*
by (*smt* (*verit*) *Mobius-gyrocarrier-norms-embed′.norms-carrier image-cong image-image mem-Collect-eq subsetD*)
qed

lemma *of-real′-E*:
assumes *abs x* < *1*
shows *GCE.of-real′ x* = *Abs-PoincareDiscE* (*PoincareDisc.of-complex* (*cor x*))
using *assms*
by (*simp add*: *GCE.of-real′-def of-carrier-E of-complex-E-def*)

lemma *to-real′-E*:
assumes *z* ∈ *GCE.reals*
shows *GCE.to-real′ z* = *Re* (*to-complex-E z*)
using *assms*
unfolding *GCE.reals-def GCE.to-real′-def*
using *GCE.norms-carrier*
by *fastforce*

lemma *gyronorm-E-lt-1* [*simp*]:
shows *abs* (*GCE.gyronorm a*) < *1*
using *GCE.gyronorm-def to-complex to-complex-E.rep-eq* **by** *auto*

lemma *gyrocarrier′-norms-E* [*simp*]:
shows *gyrocarrier′.norms to-complex-E* = *GCE.norms*
by (*simp add*: *GCE.norms-def gyrocarrier′.norms-def gyrocarrier′E*)

lemma *gyrocarrier-norms-embed′-reals-E* [*simp*]:
shows *gyrocarrier-norms-embed′.reals to-complex-E* = *GCE.reals*
by (*simp add*: *GCE.reals-def gyrocarrier-norms-embed′.reals-def gyrocarrier-norms-embed′E of-complex-E-def*)

lemma *φreals-to-reals*:

shows φ_{ME} *‘ gyrocarrier-norms-embed′.reals to-complex-M* = *gyrocarrier-norms-embed′.reals to-complex-E*
proof−
have φ_{ME} *‘ Abs-PoincareDiscM ‘ PoincareDisc.of-complex ‘ cor ‘* {*x*. |*x*| < *1*}
=
Abs-PoincareDiscE ‘ PoincareDisc.of-complex ‘ cor ‘ {*x*. |*x*| < *1*}
proof (*transfer*, *safe*)
fix *x*::*real*
assume *abs x* < *1*
then show *2* ⊗ *PoincareDisc.of-complex* (*cor x*) ∈ (λ*x*. *x*) *‘ PoincareDisc.of-complex ‘ cor ‘* {*x*. |*x*| < *1*}
by (*smt* (*verit*, *del-insts*) *Mobius-gyrocarrier-norms-embed′.bij-reals-norms Mobius-gyrocarrier-norms-embed′-of-real′ Mobius-gyrocarrier-norms-embed.otimes-reals bij-betw-imp-surj-on image-iff mem-Collect-eq*)
next
fix *x*::*real*
assume *abs x* < *1*
then show *PoincareDisc.of-complex* (*cor x*) ∈ (⊗) *2 ‘* (λ*x*. *x*) *‘ PoincareDisc.of-complex ‘ cor ‘* {*x*. |*x*| < *1*}
by *auto* (*smt* (*z3*) *Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′.norms-carrier Mobius-gyrocarrier-norms-embed.otimes-reals Mobius-pre-gyrovector-space.double-half image-iff mem-Collect-eq of-complex-inverse subsetD*)
qed
then show *?thesis*
by *simp*
qed

interpretation *gyrocarrier-norms-embed-M*: *gyrocarrier-norms-embed to-complex-M scale-M*
proof
fix *a b*
assume *a* ∈ *gyrocarrier-norms-embed′.reals to-complex-M b* ∈ *gyrocarrier-norms-embed′.reals to-complex-M*
then show *a* ⊕ *b* ∈ *gyrocarrier-norms-embed′.reals to-complex-M*
by (*smt* (*verit*, *del-insts*) *Abs-PoincareDiscM-inverse Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′.norms-carrier Mobius-gyrocarrier-norms-embed.oplus-reals Rep-PoincareDiscM-inverse UNIV-I gyrocarrier-norms-embed′-reals-M gyroplus-PoincareDiscM-def gyroplus-PoincareDisc-def image-iff of-complex-inverse oplus-M.rep-eq subset-eq*)
next
fix *a*
assume *a* ∈ *gyrocarrier-norms-embed′.reals to-complex-M*
then show ⊖ *a* ∈ *gyrocarrier-norms-embed′.reals to-complex-M*
by (*smt* (*verit*, *del-insts*) *Abs-PoincareDiscM-inverse Mobius-gyrocarrier′.of-carrier Mobius-gyrocarrier-norms-embed′.norms-carrier Mobius-gyrocarrier-norms-embed.oinv-reals Rep-PoincareDiscM-inverse UNIV-I gyrocarrier-norms-embed′-reals-M gyroinv-PoincareDiscM-def gyroinv-PoincareDisc-def image-iff of-complex-inverse ominus-M.rep-eq subset-eq*)
next
fix *r a*
assume *a* ∈ *gyrocarrier-norms-embed′.reals to-complex-M*

then show *scale-M r a* $\in$ *gyrocarrier-norms-embed'.reals to-complex-M*
by (*smt* (*verit*, *del-insts*) *Abs-PoincareDiscM-inverse Mobius-gyrocarrier'.of-carrier Mobius-gyrocarrier-norms-embed'.norms-carrier Mobius-gyrocarrier-norms-embed.otimes-reals Rep-PoincareDiscM-inverse UNIV-I gyrocarrier-norms-embed'-reals-M image-iff of-complex-inverse scale-M.rep-eq subset-eq*)
qed

interpretation *pre-gyrovector-space-M*: *pre-gyrovector-space to-complex-M scale-M*
proof
fix *u v a b*
show *GCM.gyroinner* (*gyr u v a*) (*gyr u v b*) = *GCM.gyroinner a b*
unfolding *GCM.gyroinner-def to-complex-M-def*
using *GCM.gyroinner-def gyr-M.rep-eq gyr-PoincareDiscM-def inner-p.rep-eq moebius-gyroauto to-complex-M.rep-eq to-complex-M-def*
by *force*
next
fix *a*
show *scale-M 1 a* = *a*
by (*metis Mobius-pre-gyrovector-space.scale-1 Rep-PoincareDiscM-inject scale-M.rep-eq*)
next
fix *r1 r2 a*
show *scale-M* (*r1* + *r2*) *a* = *scale-M r1 a* $\oplus$ *scale-M r2 a*
by (*metis Rep-PoincareDiscM-inject gyroplus-PoincareDiscM-def oplus-M.rep-eq otimes-oplus-m-distrib scale-M.rep-eq*)
next
fix *r1 r2 a*
show *scale-M* (*r1* $*$ *r2*) *a* = *scale-M r1* (*scale-M r2 a*)
by (*metis Rep-PoincareDiscM-inject otimes-assoc scale-M.rep-eq*)
next
fix *r*::*real* **and** *a*::*PoincareDiscM*
assume $r \neq 0$ $a \neq 0_g$
then show *to-complex-M* (*scale-M* $|r|$ *a*) $/_R$ *GCM.gyronorm* (*scale-M r a*) =
to-complex-M a $/_R$ *GCM.gyronorm a*
using *GCM.gyroinner-def to-complex-M-def*
by (*metis Abs-PoincareDiscM-cases Abs-PoincareDiscM-inverse GCM.gyronorm-def GCM.to-carrier-zero Mobius-gyrocarrier'.gyronorm-def Mobius-gyrocarrier'.to-carrier-zero Mobius-pre-gyrovector-space.scale-prop1 scale-M.rep-eq to-complex-M.rep-eq to-complex-inverse*)
next
fix *u v r a*
show *gyr u v* (*scale-M r a*) = *scale-M r* (*gyr u v a*)
by (*metis Mobius-pre-gyrovector-space.gyroauto-property Rep-PoincareDiscM-inject gyr-M.rep-eq gyr-PoincareDiscM-def gyr-PoincareDisc-def scale-M.rep-eq*)
next
fix *r1 r2 v*
show *gyr* (*scale-M r1 v*) (*scale-M r2 v*) = *id*
by (*metis Rep-PoincareDiscM-inject eq-id-iff gyr-M.rep-eq gyr-PoincareDiscM-def gyr-m-gyrospace scale-M.rep-eq*)
qed

interpretation *gyrovector-space-norms-embed-M*: *gyrovector-space-norms-embed scale-M to-complex-M*
proof
  fix *r a*
  show *GCM.gyronorm* (*scale-M r a*) = *gyrocarrier-norms-embed-M.otimesR* |*r*| (*GCM.gyronorm a*)
    using *GCM.gyroinner-def to-complex-M-def GCM.gyronorm-def*
    by (*metis GCM.inj-on-of-real′ GCM.norm-in-norms Mobius-gyrocarrier′.gyronorm-def Mobius-gyrocarrier-norms-embed′-of-real′ Mobius-gyrocarrier-norms-embed.of-real′-otimesR Mobius-gyrovector-space.homogeneity gyrocarrier-norms-embed-M.of-real′-otimesR gyrocarrier-norms-embed-M.otimesR-norms gyronorm-M-lt-1 inj-onD of-real′-M scale-M.abs-eq scale-M.rep-eq to-complex-M.rep-eq*)
next
  fix *a b*
  show *GCM.gyronorm* ($a \oplus b$) $\leq$ *GCM.oplusR* (*GCM.gyronorm a*) (*GCM.gyronorm b*)
    using *to-complex-M-def GCM.gyronorm-def GCM.oplusR-def Mobius-gyrovector-space.gyrotriangle*
    by (*smt* (*z3*) *GCM.norm-in-norms Mobius-gyrocarrier′.gyronorm-def Mobius-gyrocarrier-norms-embed′-of-r to-complex-M.rep-eq GCM.norms-carrier GCM.of-real′-def Mobius-gyrocarrier-norms-embed′.gyrocarrier-norm Mobius-gyrocarrier-norms-embed′.to-real′ gyrocarrier′.to-carrier gyrocarrier′M gyrocarrier-norms-embed′.oplusR-def gyrocarrier-norms-embed-M.of-real′-oplusR gyrocarrier-norms-embed-M.oplusR-norms gyroplus-PoincareDiscM-def gyroplus-PoincareDisc-def image-eqI o-def of-complex-M-def oplus-M.rep-eq subsetD to-complex-inverse*)
qed

lemmas *bij*φ_{ME} = *Einstein-gyrogroup-iso*.φ*bij*

lemma *oplus*φ_{ME}:
  shows φ_{ME} ($u \oplus v$) = φ_{ME} $u \oplus \varphi_{ME}$ v
  unfolding *gyroplus-PoincareDiscE-def gyroplus-PoincareDiscM-def*
  apply *transfer*
  using *iso-two-me-oplus*
  by *auto*

lemma *scale*φ_{ME}:
  shows φ_{ME} (*scale-M r u*) = *scale-E r* (φ_{ME} *u*)
  by *transfer* (*metis mult.commute otimes-assoc*)

lemma *GCEoinvRMinus*:
  assumes $a \in$ *gyrocarrier′.norms to-complex-E*
  shows *GCE.oinvR a* = − *a*
proof−
  from *assms* **have** *abs a* < *1 abs* (−*a*) < *1*
    by *auto*
  have $\ominus$ (*GCE.of-real′ a*) = *GCE.of-real′* (− *a*)
    unfolding *of-real′-E*[*OF* ‹*abs a* < *1*›] *of-real′-E*[*OF* ‹*abs* (−*a*) < *1*›]
    by (*smt* (*verit*, *ccfv-SIG*) ‹|− *a*| < *1*› *gyroinv-PoincareDiscE-def map-fun-apply mem-Collect-eq norm-of-real of-complex-inverse of-real-minus ominus-E.abs-eq ominus-e-ominus-m ominus-m′-def ominus-m-def*)

then show *?thesis*
unfolding *GCE.oinvR-def*
using ‹$a \in$ *gyrocarrier'.norms to-complex-E*›
by *auto*
qed

lemma *gyronorm*φ_{ME}:
shows φ_{ME} (*GCM.of-real'* (*GCM.gyronorm a*)) = *GCE.of-real'* (*GCE.gyronorm* (φ_{ME} *a*))
unfolding *of-real'-M*[*OF gyronorm-M-lt-1*] *of-real'-E*[*OF gyronorm-E-lt-1*]
unfolding *GCM.gyronorm-def GCE.gyronorm-def*
proof *transfer*
fix *a*
show *2* $\otimes$ *PoincareDisc.of-complex* (*cor* (*cmod* (*to-complex a*))) =
PoincareDisc.of-complex (*cor* (*cmod* (*to-complex* (*2* $\otimes$ *a*))))
proof−
{
fix *a*
assume *cmod a* < *1*
moreover
have *cmod* (*double' a*) < *1*
by (*metis* ‹*cmod a* < *1*› *double.rsp eq-onp-same-args rel-fun-eq-onp-rel*)
moreover
have *cmodcmod*: *cmod* (*cor* (*cmod a*)) < *1*
using ‹*cmod a* < *1*›
by *simp*
have *double'* (*cor* (*cmod a*)) = *cor* (*cmod* (*double' a*))
using ‹*cmod a* < *1*›
unfolding *cmod-double'*[*OF* ‹*cmod a* < *1*›]
unfolding *cmodcmod double'-def*
unfolding *double'-cmod*[*OF cmodcmod*]
by (*simp add*: *scaleR-conv-of-real*)
ultimately
have *PoincareDisc.of-complex* (*double'* (*to-complex* (*PoincareDisc.of-complex* (*cor* (*cmod a*))))) =
PoincareDisc.of-complex (*cor* (*cmod* (*to-complex* (*PoincareDisc.of-complex* (*double' a*)))))
by (*simp add*: *of-complex-inverse*)
}
note ∗ = *this*
show *?thesis*
unfolding *double*[*symmetric*] *double-def*
by (*simp*, *transfer*, *simp add*: ∗)
qed
qed

interpretation *isoME''*: *gyrocarrier-isomorphism' to-complex-M to-complex-E* φ_{ME}
by *unfold-locales*

interpretation *isoME′*: *gyrocarrier-isomorphism to-complex-M to-complex-E* φ_{ME}
proof
  show *bij* φ_{ME}
    using *bij*φ_{ME} **by** *blast*
next
  fix *u v*
  show φ_{ME} $(u \oplus v) = \varphi_{ME}$ $u \oplus \varphi_{ME}$ v
    using *oplus*φ_{ME} **by** *auto*
next
  fix *a*
  show *isoME′′*.φ_R (*GCM.gyronorm a*) = *GCE.gyronorm* (φ_{ME} *a*)
    by (*simp add*: *gyronorm*φ_{ME} *isoME′′*.φ_R*-def*)
next
  fix *u v* :: *PoincareDiscM*
  assume $u \neq 0_g$ $v \neq 0_g$
  then show *inner* (*to-complex-E* (φ_{ME} *u*) $/_R$ *GCE.gyronorm* (φ_{ME} *u*))
            (*to-complex-E* (φ_{ME} *v*) $/_R$ *GCE.gyronorm* (φ_{ME} *v*)) =
              inner (*to-complex-M u* $/_R$ *GCM.gyronorm u*) (*to-complex-M v* $/_R$
GCM.gyronorm v)
    unfolding *GCM.gyronorm-def GCE.gyronorm-def gyrozero-PoincareDiscM-def*
  proof *transfer*
    fix *u v*
    assume $u \neq 0_m$ $v \neq 0_m$
    then show *inner* (*to-complex* ($2 \otimes u$) $/_R$ *cmod* (*to-complex* ($2 \otimes u$)))
            (*to-complex* ($2 \otimes v$) $/_R$ *cmod* (*to-complex* ($2 \otimes v$))) =
            inner (*to-complex u* $/_R$ *cmod* (*to-complex u*)) (*to-complex v* $/_R$ *cmod*
(*to-complex v*))
      unfolding *double*[*symmetric*]
    proof *transfer*
      fix *u v*
      assume *cmod u* < *1* *u* ≠ *ozero-m′* *cmod v* < *1* *v* ≠ *ozero-m′*
      have $(2\ /\ (1 + (cmod\ u)^2)) *_R u /_R (2 * cmod\ u\ /\ (1 + (cmod\ u)^2)) = u$
$/_R$ *cmod u*
        by (*smt* (*verit*, *best*) ‹*cmod u* < *1*› *cmod-double′ divide-divide-eq-right dou-ble′-cmod double′-def inverse-eq-divide nonzero-mult-div-cancel-left norm-ge-zero norm-power norm-scaleR scaleR-scaleR times-divide-eq-left zero-less-divide-iff*)
      moreover
      have $(2\ /\ (1 + (cmod\ v)^2)) *_R v /_R (2 * cmod\ v\ /\ (1 + (cmod\ v)^2)) = v$
$/_R$ *cmod v*
        by (*smt* (*verit*, *best*) ‹*cmod v* < *1*› *cmod-double′ divide-divide-eq-right dou-ble′-cmod double′-def inverse-eq-divide nonzero-mult-div-cancel-left norm-ge-zero norm-power norm-scaleR scaleR-scaleR times-divide-eq-left zero-less-divide-iff*)
      ultimately
      show *inner* (*double′ u* $/_R$ *cmod* (*double′ u*)) (*double′ v* $/_R$ *cmod* (*double′ v*))
=
          inner (*u* $/_R$ *cmod u*) (*v* $/_R$ *cmod v*)
        unfolding *cmod-double′*[*OF* ‹*cmod u* < *1*›] *cmod-double′*[*OF* ‹*cmod v* < *1*›]
        unfolding *double′-def*
        unfolding *double′-cmod*[*OF* ‹*cmod u* < *1*›] *double′-cmod*[*OF* ‹*cmod v* < *1*›]

by *metis*
qed
qed
qed

interpretation *PGVME*: *pre-gyrovector-space-isomorphism to-complex-M to-complex-E scale-M scale-E* φ_{ME}
proof
fix *r u*
show φ_{ME} (*scale-M r u*) = *scale-E r* (φ_{ME} *u*)
by (*simp add*: *scale*φ_{ME})
qed

interpretation *isoME-norms-embed*′: *gyrocarrier-isomorphism-norms-embed*′ *to-complex-M to-complex-E scale-M scale-E* φ_{ME}
..

interpretation *isoME-norms-embed*: *gyrocarrier-isomorphism-norms-embed to-complex-M to-complex-E scale-M scale-E* φ_{ME}
by (*smt* (*verit*, *del-insts*) *GCE.to-real*′*-norm GCEoinvRMinus* φ*reals-to-reals bij*φ_{ME} *gyrocarrier-isomorphism-norms-embed*′.φ_R*-def gyrocarrier-isomorphism-norms-embed.intro gyrocarrier-isomorphism-norms-embed-axioms-def gyronorm*φ_{ME} *isoME-norms-embed*′*.gyrocarrier-isomorphis oplus*φ_{ME} *scale*φ_{ME})

interpretation *isoME*′: *gyrovector-space-isomorphism*′ *to-complex-M to-complex-E scale-M scale-E* φ_{ME}
by (*meson PGVME.pre-gyrovector-space-isomorphism-axioms* φ*reals-to-reals gyrovector-space-isomorphism*′*.intro gyrovector-space-isomorphism*′*-axioms-def gyrovector-space-norms-embed-M.gyrovector-space-norms-embed-axioms isoME-norms-embed.GV*′*.gyrocarrier-norms-*

interpretation *isoME*: *gyrovector-space-isomorphism to-complex-M to-complex-E scale-M scale-E* φ_{ME}
proof
fix *a b*
assume $a \in$ *gyrocarrier*′*.norms to-complex-M* $b \in$ *gyrocarrier*′*.norms to-complex-M* $0 \le a$ $a \le b$

then have $a < 1$ $b < 1$
unfolding *gyrocarrier*′*-norms-M*
by *auto*

{
fix *x*
assume $x \in$ *GCM.norms* $x \ge 0$
then have *abs* $x < 1$
by *simp*

```
    have GCM.of-real′ x ∈ GCM.reals
      unfolding of-real′-M[OF ‹abs x < 1›]
      using ‹abs x < 1›
      by auto
    then have ∗: φ_ME (GCM.of-real′ x) ∈ GCE.reals
    using φreals-to-reals gyrocarrier-norms-embed′-reals-E gyrocarrier-norms-embed′-reals-M
image-eqI
      by blast

    have GCE.to-real′ (φ_ME (GCM.of-real′ x)) = tanh (2 ∗ artanh x)
      using ‹abs x < 1› ‹0 ≤ x›
    proof (subst to-real′-E[OF ∗], subst of-real′-M[OF ‹abs x < 1›], transfer)
      fix x :: real
      assume abs x < 1 0 ≤ x
      then have ∗: otimes′ 2 (cor x) ∈ {z. cmod z < 1}
        using cmod-otimes′ cmod-otimes′-k by auto
      moreover
      have Im (otimes′ 2 (cor x)) = 0
        by (simp add: otimes′-def)
      ultimately
       show Re (to-complex (2 ⊗ PoincareDisc.of-complex (cor x))) = tanh (2 ∗
artanh x)
        unfolding otimes-def
         using of-complex-inverse[OF ∗] of-complex-inverse[of x] ‹abs x < 1› ‹0 ≤
x›
        using otimes′-k-tanh[of cor x 2]
           by (smt (verit, ccfv-SIG) Mobius-gyrocarrier′.of-carrier artanh-nonneg
cmod-otimes′ eq-onp-same-args mem-Collect-eq norm-of-real norm-p.abs-eq otimes.rep-eq
otimes-def otimes-homogenity′ tanh-0 tanh-real-less-iff)
    qed
  }
  note ∗ = this
  show isoME′′.φ_R a ≤ isoME′′.φ_R b
   using ‹a ∈ gyrocarrier′.norms to-complex-M› ‹b ∈ gyrocarrier′.norms to-complex-M›
‹0 ≤ a› ‹a ≤ b› ‹a < 1› ‹b < 1› ∗[of a] ∗[of b] tanh-artanh-mono[of a b]
    unfolding isoME′′.φ_R-def
    by simp
qed

end
theory GyroVectorSpaceTrivial
  imports GyroVectorSpace
begin
```

Every group is a gyrogroup with identity gyration

```
sublocale group-add ⊆ groupGyrogroupoid: gyrogroupoid 0 (+)
  by unfold-locales

sublocale group-add ⊆ groupGyrogroup: gyrogroup 0 (+) λ x. − x λ u v x. x
```

proof
  fix *a*
  show $0 + a = a$
    by *auto*
next
  fix *a*
  show $- a + a = 0$
    by *auto*
next
  fix *a b z*
  show $a + (b + z) = a + b + z$
    by (*simp add*: *add-assoc*)
next
  fix *a b*
  show $(\lambda\ x.\ x) = (\lambda\ x.\ x)$
    by *auto*
next
  fix *a b*
  show *gyrogroupoid.gyroaut* (+) ($\lambda x.\ x$)
    unfolding *gyrogroupoid.gyroaut-def*
    by (*auto simp add*: *bij-def*)
qed

locale *gyrocarrier-trivial* = *gyrocarrier′ to-carrier* **for**
  to-carrier :: ′*a*::{*gyrocommutative-gyrogroup*, *real-inner*, *real-normed-algebra-1*}
⇒ ′*a* +
  assumes *gyr-id*: $\bigwedge\ u\ v\ x.\ (gyr::'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a)\ u\ v\ x = x$
  assumes *to-carrier-id*: $\bigwedge\ x.\ to\text{-}carrier\ x = x$
  assumes *oplus*: $\bigwedge\ x\ y::'a\ .\ x \oplus y = x + y$
  assumes *ominus*: $\bigwedge\ x::'a\ .\ \ominus\ x = -\ x$

sublocale *gyrocarrier-trivial* ⊆ *gyrocarrier to-carrier*
proof
  fix *u v a b*
  show $gyr\ u\ v\ a \cdot gyr\ u\ v\ b = a \cdot b$
    by (*simp add*: *gyr-id*)
qed

sublocale *gyrocarrier-trivial* ⊆ *pre-gyrovector-space to-carrier* ($*_R$)
proof
  fix *a*::′*a*
  show $1\ *_R\ a = a$
    by *auto*
next
  fix *r1 r2* **and** *a*::′*a*
  show $(r1 + r2)\ *_R\ a = (r1\ *_R\ a) \oplus (r2\ *_R\ a)$
    unfolding *oplus*
    by (*meson scaleR-left-distrib*)
next

fix *r1 r2* **and** $a::'a$
show $(r1 * r2) *_R a = r1 *_R r2 *_R a$
by *simp*
next
fix $u\ v\ a :: 'a$ **and** r
show $gyr\ u\ v\ (r *_R a) = r *_R gyr\ u\ v\ a$
unfolding *gyr-id*
by *simp*
next
fix *r1 r2* **and** $v :: 'a$
show $gyr\ (r1 *_R v)\ (r2 *_R v) = id$
by (*auto simp add*: *gyr-id*)
next
fix $r::real$ **and** $a::'a$
assume $r \neq 0\ a \neq 0_g$
show $to\text{-}carrier\ (|r| *_R a) /_R \langle\langle(r *_R a)\rangle\rangle = to\text{-}carrier\ a /_R \langle\langle a\rangle\rangle$
by (*simp add*: ‹$r \neq 0$› *gyronorm-def to-carrier-id*)
qed

sublocale *gyrocarrier-trivial* $\subseteq$ TG': *gyrocarrier-norms-embed′ to-carrier*
proof
show *of-real* ‘ *norms* $\subseteq$ *carrier*
unfolding *norms-def carrier-def*
by (*simp add*: *to-carrier-id*)
qed

context *gyrocarrier-trivial*
begin

lemma *norms-UNIV*:
shows *norms* = *UNIV*
unfolding *norms-def*
by (*auto*, *metis eq-abs-iff′ gyronorm-def norm-of-real to-carrier-id*)

lemma *reals-UNIV*:
shows TG'.*reals* = *of-real* ‘ *UNIV*
unfolding TG'.*reals-def norms-UNIV*
using *of-carrier to-carrier-id*
by *auto*

lemma *of-real′*:
shows TG'.*of-real′* = *of-real*
using TG'.*of-real′-def*
using *of-carrier to-carrier-id*
by *auto*

end

sublocale *gyrocarrier-trivial* $\subseteq$ *TG*: *gyrocarrier-norms-embed to-carrier* $(*_R)$

proof
  fix *a b*
  assume $a \in TG'.reals$ $b \in TG'.reals$
  then show $a \oplus b \in TG'.reals$
    unfolding *reals-UNIV oplus*
    by (*metis Reals-add Reals-def*)
next
  fix *a*
  assume $a \in TG'.reals$
  then show $\ominus\ a \in TG'.reals$
    unfolding *reals-UNIV ominus*
    by *force*
next
  fix *a r*
  assume $a \in TG'.reals$
  then show $r *_R a \in TG'.reals$
    unfolding *reals-UNIV*
    by (*metis Reals-def Reals-mult Reals-of-real scaleR-conv-of-real*)
qed

sublocale *gyrocarrier-trivial* $\subseteq$ *gyrovector-space-norms-embed* $(*_R)$ *to-carrier*
proof
  fix *r a*
  show $\langle\!\langle(r *_R a)\rangle\!\rangle = TG.otimesR\ |r|\ (\langle\!\langle a\rangle\!\rangle)$
    unfolding *gyronorm-def TG.otimesR-def of-real'*
    by (*metis TG'.to-real' UNIV-I norm-scaleR norms-UNIV of-real' of-real-mult scaleR-conv-of-real to-carrier-id*)
next
  fix *a b*
  show $\langle\!\langle a \oplus b\rangle\!\rangle \le \langle\!\langle a\rangle\!\rangle \oplus_R (\langle\!\langle b\rangle\!\rangle)$
   by (*metis TG'.oplusR-def TG'.to-real' UNIV-I gyronorm-def norm-triangle-ineq norms-UNIV of-real' of-real-add oplus to-carrier-id*)
qed

end
theory *hDistance*
  imports *MobiusGyroVectorSpace*
begin

abbreviation *distance-m-expr* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *real* **where**
  $distance\text{-}m\text{-}expr\ u\ v \equiv 1 + 2 * (cmod\ (u - v))^2\ /\ ((1 - (cmod\ u)^2) * (1 - (cmod\ v)^2))$

definition *distance-m* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *real* **where**
  $distance\text{-}m\ u\ v = arcosh\ (distance\text{-}m\text{-}expr\ u\ v)$

lemma *arcosh-artanh-lemma*:
  shows $(cmod\ (1 - cnj\ u * v))^2 - (cmod\ (u - v))^2 = (1 - (cmod\ u)^2) * (1 - (cmod\ v)^2)$

```
proof-
  have cor ((cmod (1 - cnj u * v))^2 - (cmod (u - v))^2) = cor ((1 - (cmod u)^2)
* (1 - (cmod v)^2))
    unfolding of-real-diff of-real-mult complex-norm-square
    by (simp add: field-simps)
  then show ?thesis
    using of-real-eq-iff by blast
qed

lemma distance-m-expr-ge-1:
  fixes u v :: complex
  assumes cmod u < 1 cmod v < 1
  shows distance-m-expr u v ≥ 1
proof-
  have (cmod (u-v))^2 ≥ 0
    using zero-le-power2 by blast
  moreover
  have (1 - (cmod u)^2) *(1 - (cmod v)^2) > 0
    using assms
    using cmod-def by force
  ultimately
  show ?thesis
    by simp
qed

lemma arcosh-artanh:
  fixes u v :: complex
  assumes cmod u <1 cmod v < 1
  shows arcosh (distance-m-expr u v) =
       2 * artanh (cmod ((u-v) / (1 - (cnj u)*v)))
proof-
  let ?u = 1 - (cmod u)^2 and ?v = 1 - (cmod v)^2 and ?uv = (cmod (u - v))^2

  have arcosh (distance-m-expr u v) =
       ln (distance-m-expr u v + sqrt ((distance-m-expr u v)^2 - 1))
    using arcosh-real-def[OF distance-m-expr-ge-1[OF assms]]
    by simp
  also have ... = ln (((cmod (1 - cnj u * v))^2 + 2 * cmod (1 - cnj u * v) *
cmod (u - v) + ?uv) /
                 (?u * ?v))
  proof-
    have distance-m-expr u v = (?u * ?v + 2 * ?uv) / (?u * ?v)
    proof-
      have ?u ≠ 0 ?v ≠ 0
        using assms
      by (metis abs-norm-cancel order-less-irrefl real-sqrt-abs real-sqrt-one right-minus-eq)+

      then show ?thesis
        by (simp add: add-divide-distrib)
```

qed
then have $*$: *distance-m-expr u v* = ((*cmod* (*1* − *cnj u* ∗ *v*))^2 + *?uv*) / (*?u* ∗ *?v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *arcosh-artanh-lemma*)

have *sqrt* ((*distance-m-expr u v*)^2 − *1*) =
sqrt (*4* ∗ (*?uv* / (*?u* ∗ *?v*)) + *4* ∗ (*?uv* / (*?u* ∗ *?v*))^2)
by (*smt* (*verit*, *best*) *add-divide-distrib four-x-squared inner-real-def one-power2 power2-diff real-inner-1-right*)
also have ... = *sqrt* (*4* ∗ *?uv* ∗ (*1* / (*?u* ∗ *?v*) + (*cmod* (*u* − *v*) / (*?u* ∗ *?v*))^2)) (**is** *?lhs* = *sqrt* (*4* ∗ *?A* ∗ *?B*))
by (*simp add*: *field-simps*)
also have ... = *sqrt* (*4* ∗ *?uv* ∗ (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)^2)
proof−
have *?B* = (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)^2
by (*simp add*: *power-divide power2-eq-square add-divide-distrib*)
then show *?thesis*
by *simp*
qed
also have ... = *2* ∗ *cmod* (*u* − *v*) ∗ *sqrt* (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *four-x-squared mult-nonneg-nonneg norm-not-less-zero one-power2 power-mono real-root-divide real-root-mult real-sqrt-unique sqrt-def*)
also have ... = *2* ∗ *cmod* (*u* − *v*) / (*?u* ∗ *?v*) ∗ *sqrt* (*?u* ∗ *?v* + *?uv*)
by *simp*
finally have ∗∗: *sqrt* ((*distance-m-expr u v*)^2 − *1*) =
2 ∗ *cmod* (*u* − *v*) ∗ *cmod* (*1* − *cnj u* ∗ *v*) / (*?u* ∗ *?v*)
by (*smt* (*verit*, *del-insts*) *arcosh-artanh-lemma norm-ge-zero real-sqrt-abs times-divide-eq-left*)

show *?thesis*
using ∗ ∗∗
by (*smt* (*verit*, *best*) *add-divide-distrib power2-sum*)
qed
also have ... = *ln* ((*1* + *cmod* (*u* − *v*) / *cmod* (*1* − *cnj u* ∗ *v*)) /
(*1* − *cmod* (*u* − *v*) / *cmod* (*1* − *cnj u* ∗ *v*))) (**is** *?lhs* = *ln* (*?nom* / *?den*))
proof−
have ∗: *?nom* = (*cmod* (*u* − *v*) + *cmod* (*1* − *cnj u* ∗ *v*)) / *cmod* (*1* − *cnj u* ∗ *v*)
using *assms*
by (*metis* (*no-types*, *opaque-lifting*) *add-diff-cancel-left′ add-divide-distrib complex-mod-cnj diff-diff-eq2 diff-zero divide-self-if mult-closed-for-unit-disc norm-eq-zero norm-one order-less-irrefl*)
then have ∗∗: *?den* = (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*)) / *cmod* (*1* − *cnj u* ∗ *v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *add-divide-distrib*)

have *?nom* / *?den* =
(*cmod* (*u* − *v*) + *cmod* (*1* − *cnj u* ∗ *v*)) / (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*))
using ∗ ∗∗
by *force*
also have . . . = (*cmod* (*1* − *cnj u* ∗ *v*) + *cmod* (*u* − *v*))^2 / (*?u* ∗ *?v*) (**is** *?lhs* = *?rhs*)
proof−
let *?e* = *cmod* (*1* − *cnj u* ∗ *v*) + *cmod* (*u* − *v*)
have *?lhs* = *?lhs* ∗ *?e*/*?e*
by *fastforce*
moreover
have (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*)) ∗ *?e* = *?u* ∗ *?v*
using *arcosh-artanh-lemma*
by (*simp add*: *mult.commute power2-eq-square square-diff-square-factored*)
ultimately
show *?thesis*
by (*simp add*: *power2-eq-square*)
qed
finally
show *?thesis*
by (*simp add*: *power2-eq-square field-simps*)
qed
finally show *?thesis*
unfolding *artanh-def*
by (*simp add*: *norm-divide*)
qed

definition *distance-m-gyro* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *real* **where**
distance-m-gyro u v = *2* ∗ *artanh* (*Mobius-pre-gyrovector-space.distance u v*)

lemma *distance-equiv*:
shows *distance-m-gyro u v* = *distance-m* (*to-complex u*) (*to-complex v*)
proof−
have (《$\ominus_m$ *u* $\oplus_m$ *v*》) =
(*cmod* ((*to-complex u* − *to-complex v*) / (*1* − *cnj* (*to-complex u*) ∗ (*to-complex v*))))
by *transfer* (*simp add*: *oplus-m′-def ominus-m′-def norm-divide norm-minus-commute*)
then show *?thesis*
unfolding *distance-m-gyro-def distance-m-def Mobius-pre-gyrovector-space.distance-def gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
using *arcosh-artanh norm-lt-one norm-p.rep-eq*
by *force*
qed

definition *blaschke* **where**
blaschke a z = (*z* − *a*) / (*1* − *cnj a* ∗ *z*)

lemma

fixes *a z* :: *complex*
shows *blaschke a z* = *oplus-m′* (*ominus-m′ a*) *z*
unfolding *blaschke-def oplus-m′-def ominus-m′-def*
by (*simp add*: *minus-divide-left*)

end
theory *MobiusCollinear*
imports *MobiusGyroVectorSpace*
begin

lemma *collinear-0-proportional′*:
assumes $v \neq 0_m$
shows *Mobius-pre-gyrovector-space.collinear* $x\ 0_m\ v \longleftrightarrow (\exists\ k{::}real.$ *to-complex* $x = k * ($*to-complex* $v))$
unfolding *Mobius-pre-gyrovector-space.collinear-def gyroplus-PoincareDisc-def gyroinv-PoincareDisc-def*
using *assms*
proof *transfer*
fix *v x*
assume *cmod v* < *1 cmod x* < *1 v* ≠ *ozero-m′*
then have *v* ≠ *0*
unfolding *ozero-m′-def*
by *simp*
have (∃ *t*. *x* = (*otimes′-k t v*) ∗ *v* / *cor* (*cmod v*)) ⟷ (∃ *k* :: *real*. *x* = *k* ∗ *v*) (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis of-real-divide times-divide-eq-left*)
next
assume *?rhs*
then obtain *k*::*real* **where** *x* = *k* ∗ *v*
by *auto*

moreover

have *abs* (*k* ∗ *cmod v*) < *1*
by (*metis* ‹*cmod x* < *1*› ‹*x* = *cor k* ∗ *v*› *abs-mult abs-norm-cancel norm-mult norm-of-real*)

have *artanh* (*cmod v*) ≠ *0*
using ‹*v* ≠ *0*›
by (*simp add*: ‹*cmod v* < *1*› *artanh-not-0*)

have ∃ *t*. (*otimes′-k t v*) / (*cmod v*) = *k*
proof−
let *?t* = *artanh*(*k* ∗ *cmod v*) / *artanh* (*cmod v*)
have *tanh* (*?t* ∗ *artanh* (*cmod v*)) = *k* ∗ *cmod v*
using ‹*artanh* (*cmod v*) ≠ *0*› *tanh-artanh*[*of k* ∗ *cmod v*] ‹*abs* (*k* ∗ *cmod v*)

< 1›
by (*simp add*: *field-simps*)
then show *?thesis*
by (*metis* ‹*cmod v* < 1› ‹$v \neq 0$› *nonzero-mult-div-cancel-right norm-zero otimes′-k-tanh zero-less-norm-iff*)
qed
ultimately
show *?lhs*
by *auto*
qed
then show ($ozero\text{-}m' = v \vee (\exists t.\ x = oplus\text{-}m'\ ozero\text{-}m'\ (otimes'\ t\ (oplus\text{-}m'\ (ominus\text{-}m'\ ozero\text{-}m')\ v)))) \longleftrightarrow$
($\exists\ k{::}real.\ x = k * v$)
using ‹$v \neq 0$›
unfolding *oplus-m′-def ozero-m′-def ominus-m′-def otimes′-def*
by *simp*
qed

lemma
assumes $v \neq 0_m$
shows *Mobius-pre-gyrovector-space.collinear* $x\ 0_m\ v \longleftrightarrow$ *to-complex* $x *$ *cnj* (*to-complex* v) $=$ *cnj* (*to-complex* x) $*$ *to-complex* v
using *Mobius-gyrocarrier′.to-carrier-zero-iff assms cnj-mix-ex-real-k collinear-0-proportional′ gyrozero-PoincareDisc-def*
by *fastforce*

lemma *collinear-0-proportional*:
shows *Mobius-pre-gyrovector-space.collinear* $x\ 0_m\ v \longleftrightarrow v = 0_m \vee (\exists\ k{::}real.$ *to-complex* $x = k *$ (*to-complex* v))
by (*metis Mobius-pre-gyrovector-space.collinear-def collinear-0-proportional′*)

lemma *to-complex-0* [*simp*]:
shows *to-complex* $0_m = 0$
by *transfer* (*simp add*: *ozero-m′-def*)

lemma *to-complex-0-iff* [*iff*]:
shows *to-complex* $x = 0 \longleftrightarrow x = 0_m$
by *transfer* (*simp add*: *ozero-m′-def*)

lemma *mobius-between-0xy*:
shows *Mobius-pre-gyrovector-space.between* $0_m\ x\ y \longleftrightarrow$
($\exists\ k{::}real.\ 0 \leq k \wedge k \leq 1 \wedge$ *to-complex* $x = k *$ *to-complex* y)
proof (*cases* $y = 0_m$)
case *True*
then show *?thesis*
using *Mobius-pre-gyrovector-space.between-xyx*[*of* $0_m\ x$]
by *auto*
next
case *False*

```
  then show ?thesis
    unfolding Mobius-pre-gyrovector-space.between-def gyroplus-PoincareDisc-def
gyrozero-PoincareDisc-def gyroinv-PoincareDisc-def
  proof (transfer)
    fix x y
    assume cmod y < 1 y ≠ ozero-m' cmod x < 1
    then have y ≠ 0
      by (simp add: ozero-m'-def)

    have (∃t≥0. t ≤ 1 ∧ x = cor (otimes'-k t y) ∗ y / cor (cmod y)) =
          (∃k≥0. k ≤ 1 ∧ x = cor k ∗ y) (is ?lhs = ?rhs)
    proof
      assume ?lhs
      then obtain t where 0 ≤ t t ≤ 1 x = (otimes'-k t y / cmod y) ∗ y
        by auto
      moreover
      have 0 ≤ otimes'-k t y / cmod y
        unfolding otimes'-k-tanh[OF ‹cmod y < 1›]
        using ‹cmod y < 1› ‹t ≥ 0› tanh-artanh-nonneg
        by auto
      moreover
      have otimes'-k t y / cmod y ≤ 1
        unfolding otimes'-k-tanh[OF ‹cmod y < 1›]
        using ‹cmod y < 1› ‹y ≠ 0› artanh-nonneg ‹t ≤ 1›
          by (smt (verit, best) divide-le-eq-1 mult-le-cancel-right2 norm-le-zero-iff
strict-mono-less-eq tanh-artanh tanh-real-strict-mono)
      ultimately
      show ?rhs
        by auto
    next
      assume ?rhs
      then obtain k::real where x = k ∗ y
        by auto

      moreover

      have abs (k ∗ cmod y) < 1
       by (metis ‹cmod x < 1› ‹x = cor k ∗ y› abs-mult abs-norm-cancel norm-mult
norm-of-real)

      have artanh (cmod y) ≠ 0
        using ‹y ≠ 0›
        by (simp add: ‹cmod y < 1› artanh-not-0)

      have ∃ t. 0 ≤ t ∧ t ≤ 1 ∧ (otimes'-k t y) / (cmod y) = k
      proof−
        let ?t = artanh(k ∗ cmod y) / artanh (cmod y)
        have tanh (?t ∗ artanh (cmod y)) = k ∗ cmod y
          using ‹artanh (cmod y) ≠ 0› tanh-artanh[of k ∗ cmod y] ‹abs (k ∗ cmod
```

```
y) < 1›
          by (simp add: field-simps)
        moreover
        have ?t ≥ 0
          using ‹∃k≥0. k ≤ 1 ∧ x = cor k ∗ y› ‹cmod y < 1› ‹x = cor k ∗ y› ‹y
≠ 0›
              by (smt (verit, ccfv-SIG) artanh-nonneg calculation divide-eq-0-iff
mult-right-cancel norm-le-zero-iff of-real-eq-iff tanh-real-nonneg-iff zero-le-mult-iff)
        moreover
        have ?t ≤ 1
          using ‹∃k≥0. k ≤ 1 ∧ x = cor k ∗ y› ‹cmod y < 1› ‹x = cor k ∗ y› ‹y
≠ 0›
          by (smt (verit, ccfv-SIG) artanh-monotone artanh-nonneg calculation(1)
less-divide-eq-1 nonzero-divide-eq-eq of-real-eq-iff tanh-artanh-nonneg zero-less-norm-iff)
        ultimately show ?thesis
          by (metis ‹cmod y < 1› ‹y ≠ 0› nonzero-mult-div-cancel-right norm-zero
otimes′-k-tanh zero-less-norm-iff)
      qed
      ultimately
      show ?lhs
        by auto
    qed
    then show (∃t≥0. t ≤ 1 ∧
                    x = oplus-m′ ozero-m′ (otimes′ t (oplus-m′ (ominus-m′ ozero-m′)
y))) =
              (∃k≥0. k ≤ 1 ∧ x = cor k ∗ y)
      using ‹y ≠ 0›
      unfolding ozero-m′-def oplus-m′-def ominus-m′-def otimes′-def
      by simp
  qed
qed

end
theory MobiusGeometry
  imports MobiusGyroVectorSpace
begin

lemma mobius-collinear-u0v′:
  assumes v ≠ 0m
  shows Mobius-pre-gyrovector-space.collinear u 0m v ⟷ (∃ k::real. to-complex
u = k ∗ (to-complex v))
   unfolding Mobius-pre-gyrovector-space.collinear-def gyroplus-PoincareDisc-def
gyroinv-PoincareDisc-def
  using assms
proof transfer
  fix v u
  assume cmod v < 1 cmod u < 1 v ≠ ozero-m′
  then have v ≠ 0
    unfolding ozero-m′-def
```

```
    by simp
  have (∃t. u = (otimes'-k t v) ∗ v / cor (cmod v)) ⟷ (∃k :: real. u = k ∗ v)
(is ?lhs ⟷ ?rhs)
  proof
    assume ?lhs
    then show ?rhs
      by (metis of-real-divide times-divide-eq-left)
  next
    assume ?rhs
    then obtain k::real where u = k ∗ v
      by auto

    moreover

    have abs (k ∗ cmod v) < 1
     by (metis ‹cmod u < 1› ‹u = cor k ∗ v› abs-mult abs-norm-cancel norm-mult
norm-of-real)

    have artanh (cmod v) ≠ 0
      using ‹v ≠ 0›
      by (simp add: ‹cmod v < 1› artanh-not-0)

    have ∃ t. (otimes'-k t v) / (cmod v) = k
    proof−
      let ?t = artanh(k ∗ cmod v) / artanh (cmod v)
      have tanh (?t ∗ artanh (cmod v)) = k ∗ cmod v
        using ‹artanh (cmod v) ≠ 0› tanh-artanh[of k ∗ cmod v] ‹abs (k ∗ cmod v)
< 1›
        by (simp add: field-simps)
      then show ?thesis
         by (metis ‹cmod v < 1› ‹v ≠ 0› nonzero-mult-div-cancel-right norm-zero
otimes'-k-tanh zero-less-norm-iff)
    qed
    ultimately
    show ?lhs
      by auto
  qed
  then show (ozero-m' = v ∨ (∃t. u = oplus-m' ozero-m' (otimes' t (oplus-m'
(ominus-m' ozero-m') v)))) ⟷
        (∃ k::real. u = k ∗ v)
    using ‹v ≠ 0›
    unfolding oplus-m'-def ozero-m'-def ominus-m'-def otimes'-def
    by simp
qed

lemma mobius-collinear-u0v:
  shows Mobius-pre-gyrovector-space.collinear x 0m v ⟷
         v = 0m ∨ (∃ k::real. to-complex x = k ∗ (to-complex v))
  by (metis Mobius-pre-gyrovector-space.collinear-def mobius-collinear-u0v')
```

```
lemma mobius-between-0uv:
  shows Mobius-pre-gyrovector-space.between 0m u v ⟷
        (∃ k::real. 0 ≤ k ∧ k ≤ 1 ∧ to-complex u = k ∗ to-complex v)
proof (cases v = 0m)
  case True
  then show ?thesis
    using Mobius-pre-gyrovector-space.between-xyx[of 0m u]
    by auto
next
  case False
  then show ?thesis
     unfolding Mobius-pre-gyrovector-space.between-def gyroplus-PoincareDisc-def
gyrozero-PoincareDisc-def gyroinv-PoincareDisc-def
  proof (transfer)
    fix x y
    assume cmod y < 1 y ≠ ozero-m′ cmod x < 1
    then have y ≠ 0
      by (simp add: ozero-m′-def)

    have (∃ t≥0. t ≤ 1 ∧ x = cor (otimes′-k t y) ∗ y / cor (cmod y)) =
          (∃ k≥0. k ≤ 1 ∧ x = cor k ∗ y) (is ?lhs = ?rhs)
    proof
      assume ?lhs
      then obtain t where 0 ≤ t t ≤ 1 x = (otimes′-k t y / cmod y) ∗ y
        by auto
      moreover
      have 0 ≤ otimes′-k t y / cmod y
        unfolding otimes′-k-tanh[OF ‹cmod y < 1›]
        using ‹cmod y < 1› ‹t ≥ 0› tanh-artanh-nonneg
        by auto
      moreover
      have otimes′-k t y / cmod y ≤ 1
        unfolding otimes′-k-tanh[OF ‹cmod y < 1›]
        using ‹cmod y < 1› ‹y ≠ 0› artanh-nonneg ‹t ≤ 1›
          by (smt (verit, best) divide-le-eq-1 mult-le-cancel-right2 norm-le-zero-iff
strict-mono-less-eq tanh-artanh tanh-real-strict-mono)
      ultimately
      show ?rhs
        by auto
    next
      assume ?rhs
      then obtain k::real where x = k ∗ y
        by auto

      moreover

      have abs (k ∗ cmod y) < 1
       by (metis ‹cmod x < 1› ‹x = cor k ∗ y› abs-mult abs-norm-cancel norm-mult
```

```
norm-of-real)

      have artanh (cmod y) ≠ 0
        using ‹y ≠ 0›
        by (simp add: ‹cmod y < 1› artanh-not-0)

      have ∃ t. 0 ≤ t ∧ t ≤ 1 ∧ (otimes'-k t y) / (cmod y) = k
      proof−
        let ?t = artanh(k ∗ cmod y) / artanh (cmod y)
        have tanh (?t ∗ artanh (cmod y)) = k ∗ cmod y
          using ‹artanh (cmod y) ≠ 0› tanh-artanh[of k ∗ cmod y] ‹abs (k ∗ cmod
y) < 1›
          by (simp add: field-simps)
        moreover
        have ?t ≥ 0
          using ‹∃ k≥0. k ≤ 1 ∧ x = cor k ∗ y› ‹cmod y < 1› ‹x = cor k ∗ y› ‹y
≠ 0›
              by (smt (verit, ccfv-SIG) artanh-nonneg calculation divide-eq-0-iff
mult-right-cancel norm-le-zero-iff of-real-eq-iff tanh-real-nonneg-iff zero-le-mult-iff)
        moreover
        have ?t ≤ 1
          using ‹∃ k≥0. k ≤ 1 ∧ x = cor k ∗ y› ‹cmod y < 1› ‹x = cor k ∗ y› ‹y
≠ 0›
          by (smt (verit, ccfv-SIG) artanh-monotone artanh-nonneg calculation(1)
less-divide-eq-1 nonzero-divide-eq-eq of-real-eq-iff tanh-artanh-nonneg zero-less-norm-iff)
        ultimately show ?thesis
          by (metis ‹cmod y < 1› ‹y ≠ 0› nonzero-mult-div-cancel-right norm-zero
otimes'-k-tanh zero-less-norm-iff)
      qed
      ultimately
      show ?lhs
        by auto
    qed
    then show (∃ t≥0. t ≤ 1 ∧
                x = oplus-m' ozero-m' (otimes' t (oplus-m' (ominus-m' ozero-m')
y))) =
             (∃ k≥0. k ≤ 1 ∧ x = cor k ∗ y)
      using ‹y ≠ 0›
      unfolding ozero-m'-def oplus-m'-def ominus-m'-def otimes'-def
      by simp
  qed
qed



abbreviation distance-m-expr :: complex ⇒ complex ⇒ real where
  distance-m-expr u v ≡ 1 + 2 ∗ (cmod (u − v))^2 / ((1 − (cmod u)^2) ∗ (1 −
(cmod v)^2))
```

definition *distance-m* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *real* **where**
distance-m u v = *arcosh* (*distance-m-expr u v*)

lemma *arcosh-artanh-lemma*:
shows $(cmod\ (1 - cnj\ u * v))^2 - (cmod\ (u - v))^2 = (1 - (cmod\ u)^2) * (1 - (cmod\ v)^2)$
proof−
have $cor\ ((cmod\ (1 - cnj\ u * v))^2 - (cmod\ (u - v))^2) = cor\ ((1 - (cmod\ u)^2) * (1 - (cmod\ v)^2))$
unfolding *of-real-diff of-real-mult complex-norm-square*
by (*simp add*: *field-simps*)
then show *?thesis*
using *of-real-eq-iff* **by** *blast*
qed

lemma *distance-m-expr-ge-1*:
fixes *u v* :: *complex*
assumes $cmod\ u < 1$ $cmod\ v < 1$
shows *distance-m-expr u v* ≥ 1
proof−
have $(cmod\ (u{-}v))^2 \geq 0$
using *zero-le-power2* **by** *blast*
moreover
have $(1 - (cmod\ u)^2) *(1 - (cmod\ v)^2) > 0$
using *assms*
using *cmod-def* **by** *force*
ultimately
show *?thesis*
by *simp*
qed

lemma *arcosh-artanh*:
fixes *u v* :: *complex*
assumes $cmod\ u < 1$ $cmod\ v < 1$
shows *arcosh* (*distance-m-expr u v*) =
$2 * artanh\ (cmod\ ((u{-}v) / (1 - (cnj\ u){*}v)))$
proof−
let $?u = 1 - (cmod\ u)^2$ **and** $?v = 1 - (cmod\ v)^2$ **and** $?uv = (cmod\ (u - v))^2$

have *arcosh* (*distance-m-expr u v*) =
$ln\ (distance\text{-}m\text{-}expr\ u\ v + sqrt\ ((distance\text{-}m\text{-}expr\ u\ v)^2 - 1))$
using *arcosh-real-def*[*OF distance-m-expr-ge-1*[*OF assms*]]
by *simp*
also have $\ldots = ln\ (((cmod\ (1 - cnj\ u * v))^2 + 2 * cmod\ (1 - cnj\ u * v) * cmod\ (u - v) + ?uv) /$
$(?u * ?v))$
proof−
have *distance-m-expr u v* $= (?u * ?v + 2 * ?uv) / (?u * ?v)$
proof−

have *?u* ≠ *0* *?v* ≠ *0*
using *assms*
by (*metis abs-norm-cancel order-less-irrefl real-sqrt-abs real-sqrt-one right-minus-eq*)+

then show *?thesis*
by (*simp add*: *add-divide-distrib*)
qed
then have ∗: *distance-m-expr u v* = ((*cmod* (*1* − *cnj u* ∗ *v*))^2 + *?uv*) / (*?u* ∗ *?v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *arcosh-artanh-lemma*)

have *sqrt* ((*distance-m-expr u v*)^2 − *1*) =
sqrt (*4* ∗ (*?uv* / (*?u* ∗ *?v*)) + *4* ∗ (*?uv* / (*?u* ∗ *?v*))^2)
by (*smt* (*verit*, *best*) *add-divide-distrib four-x-squared inner-real-def one-power2 power2-diff real-inner-1-right*)
also have . . . = *sqrt* (*4* ∗ *?uv* ∗ (*1* / (*?u* ∗ *?v*) + (*cmod* (*u* − *v*) / (*?u* ∗ *?v*))^2)) (**is** *?lhs* = *sqrt* (*4* ∗ *?A* ∗ *?B*))
by (*simp add*: *field-simps*)
also have . . . = *sqrt* (*4* ∗ *?uv* ∗ (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)^2)
proof−
have *?B* = (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)^2
by (*simp add*: *power-divide power2-eq-square add-divide-distrib*)
then show *?thesis*
by *simp*
qed
also have . . . = *2* ∗ *cmod* (*u* − *v*) ∗ *sqrt* (*?u* ∗ *?v* + *?uv*) / (*?u* ∗ *?v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *four-x-squared mult-nonneg-nonneg norm-not-less-zero one-power2 power-mono real-root-divide real-root-mult real-sqrt-unique sqrt-def*)
also have ... = *2* ∗ *cmod* (*u* − *v*) / (*?u* ∗ *?v*) ∗ *sqrt* (*?u* ∗ *?v* + *?uv*)
by *simp*
finally have ∗∗: *sqrt* ((*distance-m-expr u v*)^2 − *1*) =
2 ∗ *cmod* (*u* − *v*) ∗ *cmod* (*1* − *cnj u* ∗ *v*) / (*?u* ∗ *?v*)
by (*smt* (*verit*, *del-insts*) *arcosh-artanh-lemma norm-ge-zero real-sqrt-abs times-divide-eq-left*)

show *?thesis*
using ∗ ∗∗
by (*smt* (*verit*, *best*) *add-divide-distrib power2-sum*)
qed
also have . . . = *ln* ((*1* + *cmod* (*u* − *v*) / *cmod* (*1* − *cnj u* ∗ *v*)) /
(*1* − *cmod* (*u* − *v*) / *cmod* (*1* − *cnj u* ∗ *v*))) (**is** *?lhs* = *ln* (*?nom* / *?den*))
proof−
have ∗: *?nom* = (*cmod* (*u* − *v*) + *cmod* (*1* − *cnj u* ∗ *v*)) / *cmod* (*1* − *cnj u* ∗ *v*)
using *assms*
by (*metis* (*no-types*, *opaque-lifting*) *add-diff-cancel-left′ add-divide-distrib com-*

plex-mod-cnj diff-diff-eq2 diff-zero divide-self-if mult-closed-for-unit-disc norm-eq-zero norm-one order-less-irrefl)
then have ∗∗: *?den* = (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*)) / *cmod* (*1* − *cnj u* ∗ *v*)
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *add-divide-distrib*)
have *?nom* / *?den* =
(*cmod* (*u* − *v*) + *cmod* (*1* − *cnj u* ∗ *v*)) / (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*))
using ∗ ∗∗
by *force*
also have ... = (*cmod* (*1* − *cnj u* ∗ *v*) + *cmod* (*u* − *v*))^2 / (*?u* ∗ *?v*) (**is** *?lhs* = *?rhs*)
proof−
let *?e* = *cmod* (*1* − *cnj u* ∗ *v*) + *cmod* (*u* − *v*)
have *?lhs* = *?lhs* ∗ *?e*/*?e*
by *fastforce*
moreover
have (*cmod* (*1* − *cnj u* ∗ *v*) − *cmod* (*u* − *v*)) ∗ *?e* = *?u* ∗ *?v*
using *arcosh-artanh-lemma*
by (*simp add*: *mult.commute power2-eq-square square-diff-square-factored*)
ultimately
show *?thesis*
by (*simp add*: *power2-eq-square*)
qed
finally
show *?thesis*
by (*simp add*: *power2-eq-square field-simps*)
qed
finally show *?thesis*
unfolding *artanh-def*
by (*simp add*: *norm-divide*)
qed

definition *distance*$_m$ :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *real* **where**
distance$_m$ *u v* = *2* ∗ *artanh* (*Mobius-pre-gyrovector-space.distance u v*)

lemma *distance*$_m$*-equiv*:
shows *distance*$_m$ *u v* = *distance-m* (*to-complex u*) (*to-complex v*)
proof−
have (⟪$\ominus_m$ *u* $\oplus_m$ *v*⟫) =
(*cmod* ((*to-complex u* − *to-complex v*) / (*1* − *cnj* (*to-complex u*) ∗ (*to-complex v*))))
by *transfer* (*simp add*: *oplus-m′-def ominus-m′-def norm-divide norm-minus-commute*)
then show *?thesis*
unfolding *distance*$_m$*-def distance-m-def Mobius-pre-gyrovector-space.distance-def gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
using *arcosh-artanh norm-lt-one norm-p.rep-eq*
by *force*

qed

definition $cong_m$ :: *PoincareDisc* $\Rightarrow$ *PoincareDisc* $\Rightarrow$ *PoincareDisc* $\Rightarrow$ *Poincare-Disc* $\Rightarrow$ *bool* **where**
  $cong_m\ a\ b\ c\ d \longleftrightarrow distance_m\ a\ b = distance_m\ c\ d$

end
theory *TarskiIsomorphism*
  imports *Poincare-Disc.Tarski*
begin

locale *TarskiAbsoluteIso* = *TarskiAbsolute* +
  fixes φ :: $'a \Rightarrow 'b$
  fixes $cong'$ :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$
  fixes $betw'$ :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$
  assumes φbij: $bij\ \varphi$
  assumes $\varphi cong$: $\bigwedge x\ y\ z\ w.\ cong'\ (\varphi\ x)\ (\varphi\ y)\ (\varphi\ z)\ (\varphi\ w) \longleftrightarrow cong\ x\ y\ z\ w$
  assumes $\varphi betw$: $\bigwedge x\ y\ z.\ betw'\ (\varphi\ x)\ (\varphi\ y)\ (\varphi\ z) \longleftrightarrow betw\ x\ y\ z$

sublocale *TarskiAbsoluteIso* $\subseteq$ *TA*: *TarskiAbsolute* $cong'\ betw'$
proof
  fix $x\ y$
  show $cong'\ x\ y\ y\ x$
    by (*smt* (*verit*) $\varphi cong$ φbij *bij-iff cong-reflexive*)
next
  fix $x\ y\ z\ u\ v\ w$
  show $cong'\ x\ y\ z\ u \wedge cong'\ x\ y\ v\ w \longrightarrow cong'\ z\ u\ v\ w$
    by (*smt* (*verit*) $\varphi cong$ φbij *bij-iff cong-transitive*)
next
  fix $x\ y\ z$
  show $cong'\ x\ y\ z\ z \longrightarrow x = y$
    by (*smt* (*verit*) $\varphi cong$ φbij *bij-iff cong-identity*)
next
  fix $x\ y\ a\ b$
  show $\exists\ z.\ betw'\ x\ y\ z \wedge cong'\ y\ z\ a\ b$
    by (*smt* (*verit*) $\varphi cong$ $\varphi betw$ φbij *bij-iff segment-construction*)
next
  fix $x\ y\ z\ x'\ y'\ z'\ u\ u'$
  show $x \neq y \wedge betw'\ x\ y\ z \wedge betw'\ x'\ y'\ z' \wedge cong'\ x\ y\ x'\ y' \wedge cong'\ y\ z\ y'\ z' \wedge cong'\ x\ u\ x'\ u' \wedge cong'\ y\ u\ y'\ u' \longrightarrow$
        $cong'\ z\ u\ z'\ u'$
    by (*smt* (*verit*) $\varphi cong$ $\varphi betw$ φbij *bij-iff five-segment*)
next
  fix $x\ y$
  show $betw'\ x\ y\ x \longrightarrow x = y$
    by (*metis* $\varphi betw$ φbij *betw-identity bij-pointE*)
next
  fix $x\ u\ z\ y\ v$
  show $betw'\ x\ u\ z \wedge betw'\ y\ v\ z \longrightarrow (\exists\, a.\ betw'\ u\ a\ y \wedge betw'\ x\ a\ v)$

by (*smt* (*verit*) *φbetw φbij Pasch bij-pointE*)
next
show ∃ *a b c*. ¬ *betw′ a b c* ∧ ¬ *betw′ b c a* ∧ ¬ *betw′ c a b*
by (*smt* (*verit*) *φbetw φbij lower-dimension bij-pointE*)
next
fix *x u v y z*
show *cong′ x u x v* ∧ *cong′ y u y v* ∧ *cong′ z u z v* ∧ *u* ≠ *v* ⟶ *betw′ x y z* ∨ *betw′ y z x* ∨ *betw′ z x y*
by (*smt* (*verit*) *φcong φbetw φbij upper-dimension bij-pointE*)
qed

context *TarskiAbsoluteIso*
begin
lemma *φon-line*:
shows *TA.on-line* (*φ p*) (*φ a*) (*φ b*) ⟷ *on-line p a b*
using *TA.on-line-def φbetw on-line-def*
by *presburger*

lemma *φon-ray*:
shows *TA.on-ray* (*φ p*) (*φ a*) (*φ b*) ⟷ *on-ray p a b*
using *TA.on-ray-def φbetw on-ray-def*
by *presburger*

lemma *φin-angle*:
shows *TA.in-angle* (*φ p*) (*φ a*) (*φ b*) (*φ c*) ⟷ *in-angle p a b c*
proof
assume *in-angle p a b c*
then show *TA.in-angle* (*φ p*) (*φ a*) (*φ b*) (*φ c*)
by (*smt* (*verit*, *best*) *TA.in-angle-def in-angle-def φbetw φbij φon-ray bij-is-inj inj-eq*)
next
assume *TA.in-angle* (*φ p*) (*φ a*) (*φ b*) (*φ c*)
then obtain *x* **where** *φ b* ≠ *φ a* ∧ *φ b* ≠ *φ c* ∧ *φ p* ≠ *φ b betw′* (*φ a*) (*φ x*) (*φ c*) ∧ *φ x* ≠ *φ a* ∧ *φ x* ≠ *φ c TA.on-ray* (*φ p*) (*φ b*) (*φ x*)
by (*smt* (*verit*, *ccfv-threshold*) *TA.in-angle-def φbij bij-pointE*)
then show *in-angle p a b c*
unfolding *in-angle-def*
using *φbetw φon-ray* **by** *auto*
qed

lemma *φray-meets-line*:
shows *TA.ray-meets-line* (*φ ra*) (*φ rb*) (*φ la*) (*φ lb*) ⟷
ray-meets-line ra rb la lb
unfolding *TA.ray-meets-line-def ray-meets-line-def*
by (*metis φbij φon-line φon-ray bij-pointE*)

end

locale *TarskiHyperbolicIso* = *TarskiHyperbolic* +

fixes φ :: $'a \Rightarrow 'b$
fixes *cong'* :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*
fixes *betw'* :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*
assumes *φbij*: *bij φ*
assumes *φcong*: $\bigwedge$ *x y z w. cong'* (*φ x*) (*φ y*) (*φ z*) (*φ w*) $\longleftrightarrow$ *cong x y z w*
assumes *φbetw*: $\bigwedge$ *x y z. betw'* (*φ x*) (*φ y*) (*φ z*) $\longleftrightarrow$ *betw x y z*

sublocale *TarskiHyperbolicIso* ⊆ *TAI*: *TarskiAbsoluteIso*
by (*simp add*: *TarskiAbsoluteIso.intro TarskiAbsoluteIso-axioms-def TarskiAbsolute-axioms φbetw φbij φcong*)

sublocale *TarskiHyperbolicIso* ⊆ *TarskiHyperbolic cong' betw'*
proof
show ∃ *a b c d t. betw' a d t* ∧ *betw' b d c* ∧ *a* ≠ *d* ∧ (∀ *x y. betw' a b x* ∧ *betw' a c y* ⟶ ¬ *betw' x t y*)
by (*smt* (*verit*) *φbetw φbij euclid-negation bij-pointE*)
next
fix *a x1 x2*
assume ¬ *TAI.TA.on-line a x1 x2*
then have ∗: ¬ *on-line* (*inv φ a*) (*inv φ x1*) (*inv φ x2*)
by (*metis φbij TAI.φon-line bij-inv-eq-iff*)
obtain *a1 a2* **where** ∗:
¬ *on-line* (*inv φ a*) (*inv φ a1*) (*inv φ a2*)
¬ *ray-meets-line* (*inv φ a*) (*inv φ a1*) (*inv φ x1*) (*inv φ x2*)
¬ *ray-meets-line* (*inv φ a*) (*inv φ a2*) (*inv φ x1*) (*inv φ x2*)
(∀ *a'. in-angle a'* (*inv φ a1*) (*inv φ a*) (*inv φ a2*) ⟶ *ray-meets-line* (*inv φ a*) *a'* (*inv φ x1*) (*inv φ x2*))
using *limiting-parallels*[*OF* ∗]
by (*smt* (*verit*, *ccfv-threshold*) *φbij bij-inv-eq-iff*)
then have ¬ *TAI.TA.on-line a a1 a2* ∧ ¬ *TAI.TA.ray-meets-line a a1 x1 x2* ∧ ¬ *TAI.TA.ray-meets-line a a2 x1 x2*
by (*metis φbij TAI.φon-line TAI.φray-meets-line bij-is-surj surj-f-inv-f*)
moreover
have ∀ *a'. TAI.TA.in-angle a' a1 a a2* ⟶ *TAI.TA.ray-meets-line a a' x1 x2*
proof *safe*
fix *a'*
assume *TAI.TA.in-angle a' a1 a a2*
then have *ray-meets-line* (*inv φ a*) (*inv φ a'*) (*inv φ x1*) (*inv φ x2*)
using ∗(*4*) *TAI.φray-meets-line TAI.φin-angle φbij bij-pointE*
by (*smt* (*verit*, *ccfv-SIG*) *bij-is-surj surj-f-inv-f*)
then show *TAI.TA.ray-meets-line a a' x1 x2*
by (*metis φbij TAI.φray-meets-line bij-is-surj surj-f-inv-f*)
qed
ultimately show ∃ *a1 a2*.
¬ *TAI.TA.on-line a a1 a2* ∧
¬ *TAI.TA.ray-meets-line a a1 x1 x2* ∧
¬ *TAI.TA.ray-meets-line a a2 x1 x2* ∧
(∀ *a'. TAI.TA.in-angle a' a1 a a2* ⟶ *TAI.TA.ray-meets-line a a' x1 x2*)
by *blast*

qed

locale *ElementaryTarskiHyperbolicIso* = *ElementaryTarskiHyperbolic* +
  fixes φ :: $'a \Rightarrow 'b$
  fixes *cong'* :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*
  fixes *betw'* :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*
  assumes φ*bij*: *bij* φ
  assumes φ*cong*: $\bigwedge$ *x y z w*. *cong'* (φ *x*) (φ *y*) (φ *z*) (φ *w*) $\longleftrightarrow$ *cong x y z w*
  assumes φ*betw*: $\bigwedge$ *x y z*. *betw'* (φ *x*) (φ *y*) (φ *z*) $\longleftrightarrow$ *betw x y z*

sublocale *ElementaryTarskiHyperbolicIso* $\subseteq$ *THI*: *TarskiHyperbolicIso*
  by (*simp add*: *TarskiHyperbolicIso.intro TarskiHyperbolicIso-axioms-def Tarski-Hyperbolic-axioms* φ*betw* φ*bij* φ*cong*)

sublocale *ElementaryTarskiHyperbolicIso* $\subseteq$ *ElementaryTarskiHyperbolic cong' betw'*
proof
  fix Φ Ψ
  assume $\exists$ *a*. $\forall$ *x y*. Φ *x* $\wedge$ Ψ *y* $\longrightarrow$ *betw' a x y*
  then have $\exists$ *a*. $\forall$ *x y*. ($\Phi \circ \varphi$) *x* $\wedge$ ($\Psi \circ \varphi$) *y* $\longrightarrow$ *betw a x y*
    by (*metis* (*no-types*, *opaque-lifting*) φ*betw* φ*bij bij-pointE comp-eq-dest-lhs*)
  then have $\exists$ *b*. $\forall$ *x y*. ($\Phi \circ \varphi$) *x* $\wedge$ ($\Psi \circ \varphi$) *y* $\longrightarrow$ *betw x b y*
    using *continuity*
    by *simp*
  then show $\exists$ *b*. $\forall$ *x y*. Φ *x* $\wedge$ Ψ *y* $\longrightarrow$ *betw' x b y*
    by (*metis* (*no-types*, *opaque-lifting*) φ*betw* φ*bij bij-pointE comp-eq-dest-lhs*)
qed

end
theory *MobiusGyroTarski*
imports *MobiusGeometry TarskiIsomorphism Poincare-Disc.Poincare-Tarski*
begin

This theory depends on the following AFP entries:
https://www.isa-afp.org/entries/Poincare_Disc.html https://www.isa-afp.org/entries/Complex_Geometry.html

They must be downloaded in order to check this theory.

The following lemmas can be moved to the cited AFP entries.

lemma *eqArgLessCmod*:
  assumes $u \neq 0$ $v \neq 0$
  shows *Arg u* = *Arg v* $\wedge$ *cmod u* $\leq$ *cmod v* $\longleftrightarrow$ ($\exists$ *k*. $k \geq 0 \wedge k \leq 1 \wedge u =$ *cor* $k * v$)
proof
  assume *Arg u* = *Arg v* $\wedge$ *cmod u* $\leq$ *cmod v*
  then show $\exists$ *k*. $k \geq 0 \wedge k \leq 1 \wedge u =$ *cor* $k * v$
    using *cmod-cis*[*OF* ‹$u \neq 0$›] *cmod-cis*[*OF* ‹$v \neq 0$›] *assms*
    by (*rule-tac x=cmod u / cmod v* **in** *exI*)
      (*smt* (*verit*, *ccfv-threshold*) *divide-le-eq-1-pos divide-nonneg-nonneg mult.assoc mult-cancel-right2 nonzero-eq-divide-eq norm-ge-zero of-real-divide of-real-eq-1-iff*)

next
  assume $(\exists k.\ k \geq 0 \wedge k \leq 1 \wedge u = cor\ k * v)$
  then show $Arg\ u = Arg\ v \wedge cmod\ u \leq cmod\ v$
   by (*metis abs-of-nonneg arg-mult-real-positive assms*(*1*) *mult.commute mult-eq-0-iff mult-left-le norm-ge-zero norm-mult norm-of-real zero-less-norm-iff*)
qed

lift-definition *p-blaschke* :: *p-point* ⇒ *p-isometry* **is** λ *a*. (*moebius-pt* (*blaschke* (*to-complex a*)))
 by (*metis blaschke-unit-disc-fix inf-notin-unit-disc of-complex-to-complex unit-disc-fix-f-moebius-pt unit-disc-iff-cmod-lt-1*)

lemma *p-between-p-isometry-pt* [*simp*]:
  shows *p-between* (*p-isometry-pt f a*) (*p-isometry-pt f b*) (*p-isometry-pt f c*) ⟷ *p-between a b c*
  by *transfer* (*auto simp add*: *unit-disc-fix-f-def*)

lemma *p-blaschke-id* [*simp*]:
  shows *p-isometry-pt* (*p-blaschke x*) *x* = *p-zero*
   by *transfer* (*metis blaschke-a-to-zero inversion-infty inversion-noteq-unit-disc less-irrefl of-complex-to-complex unit-disc-iff-cmod-lt-1 zero-in-unit-disc*)

lemma *p-between-0uv*:
  shows *p-between p-zero u v* ⟷
      $(\exists k{\geq}0.\ k \leq 1 \wedge$ *to-complex* (*Rep-p-point u*) = *cor k* ∗ *to-complex* (*Rep-p-point v*))
proof *transfer*
  fix *u v*
  assume *uv*: $u \in$ *unit-disc* $v \in$ *unit-disc*
  then show *poincare-between* 0_h *u v* =
      $(\exists k{\geq}0.\ k \leq 1 \wedge$ *to-complex u* = *cor k* ∗ *to-complex v*)
  proof (*cases* $u = 0_h \vee v = 0_h$)
    case *True*
    then show *?thesis*
     by (*metis dual-order.refl inf-notin-unit-disc linordered-nonzero-semiring-class.zero-le-one mult-cancel-left1 mult-zero-class.mult-zero-right of-complex-to-complex of-complex-zero of-real-0 poincare-between-nonstrict*(*1*) *poincare-between-sandwich to-complex-zero-zero uv*(*1*) *zero-in-unit-disc*)
  next
    case *False*
    then have *z*: $u \neq 0_h$ $v \neq 0_h$
      by *auto*
    let *?u* = *to-complex u* **and** *?v* = *to-complex v*
    have *poincare-between* 0_h *u v* ⟷ *Arg ?u* = *Arg ?v* ∧ *cmod ?u* ≤ *cmod ?v*
      using *poincare-between-0uv*[*OF uv z*]
      by (*auto simp add*: *Let-def*)
    also have ... ⟷ $(\exists k{\geq}0.\ k \leq 1 \wedge$ *?u* = *cor k* ∗ *?v*)

by (*metis False eqArgLessCmod to-complex-zero-zero unit-disc-to-complex-inj uv*(*1*) *uv*(*2*) *zero-in-unit-disc*)
finally show *?thesis*
.
qed
qed

A bijection between AFP type representing the Poincare disc (based on complex homogenous coordinates) and our type for poincare disc (based on ordinary complex numbers)

lift-definition φ :: *p-point* $\Rightarrow$ *PoincareDisc* **is** *to-complex*
by (*metis inf-notin-unit-disc of-complex-to-complex unit-disc-iff-cmod-lt-1*)

lemma *distance-m-p-dist*:
shows *distance-m* (*PoincareDisc.to-complex* (φ *x*)) (*PoincareDisc.to-complex* (φ *y*)) = *p-dist x y*
unfolding φ*.rep-eq*
proof *transfer*
fix *x y*
assume *x* $\in$ *unit-disc y* $\in$ *unit-disc*
then show *distance-m* (*Homogeneous-Coordinates.to-complex x*) (*Homogeneous-Coordinates.to-complex y*) =
poincare-distance x y
by (*simp add*: *distance-m-def poincare-distance-formula*)
qed

definition *blaschke'* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *complex* **where**
blaschke' a z = (*z* − *a*) / (*1* − *cnj a* ∗ *z*)

lemma *blaschke'-translation*:
fixes *a z* :: *complex*
shows *blaschke' a z* = *oplus-m'* (*ominus-m' a*) *z*
unfolding *blaschke'-def oplus-m'-def ominus-m'-def*
by (*simp add*: *minus-divide-left*)

lift-definition *blaschke-g* :: *PoincareDisc* $\Rightarrow$ *PoincareDisc* $\Rightarrow$ *PoincareDisc* **is** *blaschke'*
using *blaschke'-translation ominus-m'-in-disc oplus-m'-in-disc* **by** *presburger*

lemma *blaschke-translation*:
blaschke-g a z = ($\ominus_m$ *a*) $\oplus_m$ *z*
by *transfer* (*simp add*: *blaschke'-translation*)

Isomorphism between hyperbolic geometry of Poincare disc defined in AFP entry, and hyperbolic geometry in Mobius gyrovector space. Since these two are isomorphic, the geometry of Mobius gyrovector space satisfies Tarski axioms.

interpretation *MobiusGyroTarskiIso*: *ElementaryTarskiHyperbolicIso p-congruent p-between* φ $cong_m$ *Mobius-pre-gyrovector-space.between*

proof
  show *bij* φ
    unfolding *bij-def inj-def surj-def*
      by *transfer* (*metis inf-notin-unit-disc mem-Collect-eq of-complex-to-complex*
to-complex-of-complex unit-disc-iff-cmod-lt-1)
next
  fix *x y z w*
  show $cong_m$ (φ *x*) (φ *y*) (φ *z*) (φ *w*) ⟷ *p-congruent x y z w*
    unfolding $cong_m$*-def* $distance_m$*-equiv p-congruent-def*
    by (*simp add*: *distance-m-p-dist*)
next
  fix *x y z*
  show *Mobius-pre-gyrovector-space.between* (φ *x*) (φ *y*) (φ *z*) ⟷ *p-between x y*
z
  proof−
    let *?f* = λ *a*. ⊖ (φ *x*) ⊕ *a*
    let *?f′* = λ *a*. *p-isometry-pt* (*p-blaschke x*) *a*

    have ∗: ∀ *a*. *PoincareDisc.to-complex* (*?f* (φ *a*)) = *to-complex* (*Rep-p-point*
(*?f′ a*))
    unfolding *gyroplus-PoincareDisc-def gyroinv-PoincareDisc-def blaschke-translation*[*symmetric*]
    proof (*transfer*, *safe*)
      fix *x a*
      assume *x* ∈ *unit-disc a* ∈ *unit-disc*
      then have ∗: *to-complex* (*moebius-pt* (*blaschke* (*to-complex x*)) *a*) =
            ((*to-complex a* − *to-complex x*) / (*1* − *cnj* (*to-complex x*) ∗ *to-complex*
a))
        using *moebius-pt-blaschke*[*of to-complex x to-complex a*]
          by (*smt* (*verit*) *blaschke-a-to-zero complex-cnj-zero-iff diff-zero div-by-0*
div-by-1 inf-notin-unit-disc inversion-noteq-unit-disc inversion-of-complex mult-eq-0-iff
of-complex-to-complex to-complex-of-complex to-complex-zero-zero unit-disc-iff-cmod-lt-1)

        show *blaschke′* (*to-complex x*) (*to-complex a*) = *to-complex* (*moebius-pt*
(*blaschke* (*to-complex x*)) *a*)
        unfolding ∗ *blaschke′-def*
        by *simp*
    qed

    have *Mobius-pre-gyrovector-space.between* (φ *x*) (φ *y*) (φ *z*) ⟷
          Mobius-pre-gyrovector-space.between 0_m (*?f* (φ *y*)) (*?f* (φ *z*))
    by (*metis Mobius-pre-gyrovector-space.between-translate Mobius-pre-gyrovector-space.translate-def*
gyro-left-inv gyrozero-PoincareDisc-def)
    also have . . . ⟷ (∃ *k*≥*0*. *k* ≤ *1* ∧ *PoincareDisc.to-complex* (*?f* (φ *y*)) =
More-Complex.cor k ∗ *PoincareDisc.to-complex* (*?f* (φ *z*)))
      using *mobius-between-0uv*
      by *simp*
    also have . . . ⟷ (∃ *k*≥*0*. *k* ≤ *1* ∧ *to-complex* (*Rep-p-point* (*?f′ y*)) =
More-Complex.cor k ∗ *to-complex* (*Rep-p-point* (*?f′ z*)))
      using ∗

by *auto*
also have ... $\longleftrightarrow$ *p-between p-zero* (*?f′ y*) (*?f′ z*)
using *p-between-0uv*
by *blast*
also have ... $\longleftrightarrow$ *p-between* (*?f′ x*) (*?f′ y*) (*?f′ z*)
by *simp*
finally show *?thesis*
using *p-between-p-isometry-pt* **by** *blast*
qed
qed

interpretation *MobiusGyroTarski*: *ElementaryTarskiHyperbolic* $cong_m$ *Mobius-pre-gyrovector-space.between*
by (*simp add*: *MobiusGyroTarskiIso.ElementaryTarskiHyperbolic-axioms*)

end
theory *MobiusGyrotrigonometry*
imports *Main GammaFactor PoincareDisc MobiusGyroVectorSpace MoreComplex*
begin

lemma *m-gamma-h1*:
shows $\ominus_m$ *a* $\oplus_m$ *b* = *of-complex* ((*to-complex b* − *to-complex a*) / (*1* − *cnj* (*to-complex a*) ∗ *to-complex b*))
by (*metis Mobius-gyrocarrier′.of-carrier add-uminus-conv-diff complex-cnj-minus mult-minus-left ominus-m′-def ominus-m.rep-eq oplus-m′-def oplus-m.rep-eq uminus-add-conv-diff*)

lemma *m-gamma-h2*:
shows $(\langle\!\langle \ominus_m\ a \oplus_m\ b \rangle\!\rangle)^2$ =
$((\langle\!\langle b \rangle\!\rangle)^2 + (\langle\!\langle a \rangle\!\rangle)^2$ − (*to-complex a*) ∗ *cnj* (*to-complex b*) − *cnj* (*to-complex a*) ∗ (*to-complex b*)) /
(*1* − (*to-complex a*) ∗ *cnj* (*to-complex b*) − *cnj*(*to-complex a*) ∗ (*to-complex b*) + $(\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2$)
proof−
let *?a* = *to-complex a* **and** *?b* = *to-complex b*
have (*?b* − *?a*) ∗ *cnj* (*?b* − *?a*) = $(\langle\!\langle b \rangle\!\rangle)^2 + (\langle\!\langle a \rangle\!\rangle)^2$ − *?a* ∗ *cnj ?b* − *cnj ?a* ∗ *?b*
by (*simp add*: *cnj-cmod mult.commute norm-p.rep-eq right-diff-distrib′*)
moreover
have (*1* − *cnj ?a* ∗ *?b*) ∗ *cnj* (*1* − *cnj ?a* ∗ *?b*) =
1 − *?a* ∗ *cnj ?b* − *cnj ?a* ∗ *?b* + $(\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2$
by (*smt* (*verit*, *ccfv-threshold*) *complex-cnj-cnj complex-cnj-diff complex-cnj-mult complex-mod-cnj complex-norm-square diff-add-eq diff-diff-eq2 left-diff-distrib mult.right-neutral mult-1 norm-mult norm-p.rep-eq power-mult-distrib right-diff-distrib*)
moreover
have (*?b* − *?a*) ∗ *cnj* (*?b* − *?a*) /
((*1* − *cnj ?a* ∗ *?b*) ∗ *cnj* (*1* − *cnj ?a* ∗ *?b*)) =
$\langle\!\langle \ominus_m\ a \oplus_m\ b \rangle\!\rangle * \langle\!\langle \ominus_m\ a \oplus_m\ b \rangle\!\rangle$
using *m-gamma-h1*

by (*metis* (*no-types*, *lifting*) *Mobius-gyrocarrier'.gyronorm-def add.commute complex-cnj-divide complex-cnj-minus complex-norm-square mult-minus-left ominus-m'-def ominus-m.rep-eq oplus-m'-def oplus-m.rep-eq power2-eq-square times-divide-times-eq uminus-add-conv-diff*)
ultimately show *?thesis*
unfolding *power2-eq-square*
by *metis*
qed

lemma *m-gamma-h3*:
shows $1 - (\langle\!\langle \ominus_m\ a \oplus_m\ b \rangle\!\rangle)^2 =$
$(1 - (\langle\!\langle b \rangle\!\rangle)^2 - (\langle\!\langle a \rangle\!\rangle)^2 + (\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2)$ /
(*1* − (*to-complex a*) ∗ *cnj* (*to-complex b*) − *cnj* (*to-complex a*) ∗ (*to-complex b*) + $(\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2$) (**is** *?lhs* = *?rhs*)
proof−
let *?a* = *to-complex a* **and** *?b* = *to-complex b*
let *?nom* = $(\langle\!\langle b \rangle\!\rangle)^2 + (\langle\!\langle a \rangle\!\rangle)^2$ − *?a* ∗ *cnj ?b* − *cnj ?a* ∗ *?b*
let *?den* = *1* − *?a* ∗ *cnj ?b* − *cnj ?a* ∗ *?b* + $(\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2$

have *?den* ≠ *0*
proof−
have *1* − *cnj ?a* ∗ *?b* ≠ *0*
by (*metis complex-mod-cnj less-irrefl mult-closed-for-unit-disc norm-lt-one norm-one norm-p.rep-eq right-minus-eq*)
moreover
have *cnj* (*1* − *cnj ?a* ∗ *?b*) ≠ *0*
using ‹*1* − *cnj ?a* ∗ *?b* ≠ *0*›
by *fastforce*
moreover
have *cnj* (*1* − *cnj ?a* ∗ *?b*) = *1* − *?a* ∗ *cnj ?b*
by *simp*
then have *?den* = (*1* − *cnj ?a* ∗ *?b*) ∗ *cnj* (*1* − *cnj ?a* ∗ *?b*)
unfolding *power2-eq-square*
using *complex-norm-square norm-p.rep-eq*
by (*simp add*: *left-diff-distrib power2-eq-square right-diff-distrib*)
ultimately
show *?thesis*
by *auto*
qed

have *?lhs* = *1* − *?nom*/*?den*
using *m-gamma-h2*
by *simp*
also have . . . = (*?den* − *?nom*) / *?den*
using ‹*?den* ≠ *0*›
by (*simp add*: *field-simps*)
also have . . . = $(1 - (\langle\!\langle b \rangle\!\rangle)^2 - (\langle\!\langle a \rangle\!\rangle)^2 + (\langle\!\langle a \rangle\!\rangle)^2 * (\langle\!\langle b \rangle\!\rangle)^2)$ / *?den*
by *force*
finally show *?thesis*

.
qed

lift-definition *gammma-factor-m* :: *PoincareDisc* ⇒ *real* (γ_m) **is** *gamma-factor*
.

lemma *m-gamma-h4*:
shows $(\gamma_m\ (\ominus_m\ a \oplus_m\ b))^2\ =$
$(1 - (\textit{to-complex}\ a) * \textit{cnj}\ (\textit{to-complex}\ b) - \textit{cnj}\ (\textit{to-complex}\ a) * (\textit{to-complex}\ b) + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2)\ /$
$(1 - (\langle\!\langle b\rangle\!\rangle)^2 - (\langle\!\langle a\rangle\!\rangle)^2 + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2)$
proof−
have $(\gamma_m\ (\ominus_m\ a \oplus_m\ b))^2\ =\ 1\ /\ (1 - (\langle\!\langle \ominus_m\ a \oplus_m\ b\rangle\!\rangle)^2)$
proof−
have $\langle\!\langle \ominus_m\ a \oplus_m\ b\rangle\!\rangle < 1$
using *norm-lt-one* **by** *auto*
then show *?thesis*
using *gamma-factor-square-norm norm-p.rep-eq*
by (*metis gammma-factor-m.rep-eq*)
qed
then show *?thesis*
using *m-gamma-h3* **by** *auto*
qed

lemma *m-gamma-equation*:
shows $(\gamma_m\ (\ominus_m\ a \oplus_m\ b))^2 = (\gamma_m\ a)^2 * (\gamma_m\ b)^2 * (1 - 2 * a \cdot b + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2)$
proof−
let $?a = \textit{to-complex}\ a$ **and** $?b = \textit{to-complex}\ b$
have $2 * a \cdot b = ?a * \textit{cnj}\ ?b + ?b * \textit{cnj}\ ?a$
using *Mobius-gyrocarrier′.gyroinner-def two-inner-cnj* **by** *force*
then have $1 - 2 * a \cdot b + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2 = (1 - ?a * \textit{cnj}\ ?b - ?b * \textit{cnj}\ ?a + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2)$
by *simp*

moreover have $(\gamma_m\ a)*(\gamma_m\ a) = 1\ /\ (1 - (\langle\!\langle a\rangle\!\rangle)^2)$
by (*metis gamma-factor-p-square-norm gammma-factor-m.rep-eq gammma-factor-p.rep-eq power2-eq-square*)

moreover have $(\gamma_m\ b)*(\gamma_m\ b) = 1\ /\ (1 - (\langle\!\langle b\rangle\!\rangle)^2)$
by (*metis gamma-factor-p-square-norm gammma-factor-m-def gammma-factor-p-def power2-eq-square*)

moreover have $(1\ /\ (1 - (\langle\!\langle a\rangle\!\rangle)^2)) * (1\ /\ (1 - (\langle\!\langle b\rangle\!\rangle)^2)) = 1\ /\ (1 - (\langle\!\langle a\rangle\!\rangle)^2 - (\langle\!\langle b\rangle\!\rangle)^2 + (\langle\!\langle a\rangle\!\rangle)^2 * (\langle\!\langle b\rangle\!\rangle)^2)$
unfolding *power2-eq-square*
by (*simp add*: *field-simps*)

ultimately show *?thesis*

using *m-gamma-h4*
unfolding *power2-eq-square*
by (*smt* (*verit*, *del-insts*) *mult.commute mult-1 of-real-1 of-real-divide of-real-eq-iff of-real-mult times-divide-eq-left*)
qed

lemma *T8-25-help1*:
assumes $A\ t \neq B\ t\ A\ t \neq C\ t\ C\ t \neq B\ t$
$a = (\langle\langle Mobius\text{-}pre\text{-}gyrovector\text{-}space.get\text{-}a\ t\rangle\rangle)^2\ b = (\langle\langle Mobius\text{-}pre\text{-}gyrovector\text{-}space.get\text{-}b\ t\rangle\rangle)^2\ c = (\langle\langle Mobius\text{-}pre\text{-}gyrovector\text{-}space.get\text{-}c\ t\rangle\rangle)^2$
shows *to-complex* ((*of-complex a*) $\oplus_m$ (*of-complex b*) $\oplus_m$ ($\ominus_m$ (*of-complex c*))) =
($a + b - c - a*b*c$) / ($1 + a*b - a*c - b*c$) (**is** *?lhs* = *?rhs*)
proof−
have ∗: *norm a* < *1 norm b* < *1 norm c* < *1*
using *assms*
by (*simp add*: *norm-geq-zero norm-lt-one power-less-one-iff*)+

have ∗∗: $1 + a*b \neq 0$
using *abs-inner-lt-1* ∗ **by** *fastforce*

have (*of-complex a*) $\oplus_m$ (*of-complex b*) = *of-complex* ((*cor a* + *cor b*) / (*1* + *cnj a* ∗ *b*))
using ∗
by (*metis* (*mono-tags*, *lifting*) *Mobius-gyrocarrier'.of-carrier Mobius-gyrocarrier'.to-carrier mem-Collect-eq norm-of-real oplus-m'-def oplus-m.rep-eq real-norm-def*)

have *?lhs* =
(((*a* + *b*) / (*1* + *cnj a* ∗ *b*)) − *c*) / (*1* − *cnj*((*a* + *b*) / (*1* + *cnj a* ∗ *b*))∗*c*)
using *Mobius-gyrocarrier'.to-carrier* ∗ *ominus-m'-def ominus-m.rep-eq oplus-m'-def oplus-m.rep-eq real-norm-def*
by *auto*
also have ... = (((*a* + *b*) / (*1* + *a* ∗ *b*)) − *c*) / (*1* − ((*a* + *b*) / (*1* + *a*∗*b*)) ∗ *c*)
by *simp*
also have ... = ((*a* + *b* − *c* − *a*∗*b*∗*c*) / (*1* + *a*∗*b*)) / ((*1* + *a*∗*b* − *a*∗*c* − *b*∗*c*) / (*1*+*a*∗*b*))
using ∗∗
by (*simp add*: *field-simps*)
also have ... = *?rhs*
using ∗∗
by *auto*
finally show *?thesis*
.
qed

lemma *T8-25-help2*:
fixes *t* :: *PoincareDisc otriangle*
assumes $(A\ t) \neq (B\ t)\ (A\ t) \neq (C\ t)\ (C\ t) \neq (B\ t)$
$a = \langle\langle Mobius\text{-}pre\text{-}gyrovector\text{-}space.get\text{-}a\ t\rangle\rangle\ b = \langle\langle Mobius\text{-}pre\text{-}gyrovector\text{-}space.get\text{-}b$

$t\rangle\!\rangle$ c = $\langle\!\langle$*Mobius-pre-gyrovector-space.get-c t*$\rangle\!\rangle$
gamma = *Mobius-pre-gyrovector-space.get-gamma t*
shows *cos gamma* = $(a^2 + b^2 - c^2 - (a*b*c)^2)$ / $(2 * a * b * (1 - c^2))$
proof−
let *?a* = *Mobius-pre-gyrovector-space.get-a t* **and** *?b* = *Mobius-pre-gyrovector-space.get-b t* **and** *?c* = *Mobius-pre-gyrovector-space.get-c t*
have $\ominus_m$ *?a* $\oplus_m$ *?b* = gyr_m ($\ominus_m$ (*C t*)) (*B t*) *?c*
unfolding *Mobius-pre-gyrovector-space.get-a-def Mobius-pre-gyrovector-space.get-b-def Mobius-pre-gyrovector-space.get-c-def*
by (*metis gyr-PoincareDisc-def gyro-translation-2a gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*)
then have $\langle\!\langle\ominus_m$ *?a* $\oplus_m$ *?b*$\rangle\!\rangle$ = $\langle\!\langle$*?c*$\rangle\!\rangle$
by (*simp add*: *mobius-gyroauto-norm*)
then have ∗: γ_m ($\ominus_m$ *?a* $\oplus_m$ *?b*) = γ_m *?c*
by (*simp add*: *gamma-factor-def gammma-factor-m.rep-eq norm-p.rep-eq*)
then have *abc*: $(\gamma_m (\ominus_m\ ?a \oplus_m\ ?b))^2 = (\gamma_m\ ?c)^2$
by *presburger*

have $\ominus_m$ (*C t*) $\oplus_m$ *A t* $\neq 0_m$
using *assms*
by (*simp add*: *Mobius-gyrogroup.gyro-equation-right*)
then have $b \neq 0$
using *assms*
unfolding *Mobius-pre-gyrovector-space.get-b-def*
using *gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
by (*simp add*: *Mobius-gyrocarrier′.gyronorm-def*)

have $\ominus_m$ (*C t*) $\oplus_m$ *B t* $\neq 0_m$
using *assms*
by (*simp add*: *Mobius-gyrogroup.gyro-equation-right*)
then have $a \neq 0$
using *assms*
unfolding *Mobius-pre-gyrovector-space.get-a-def*
using *gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
by (*simp add*: *Mobius-gyrocarrier′.gyronorm-def*)

have $1 - c^2 \neq 0$
using *assms*
by (*metis abs-norm-cancel dual-order.refl eq-iff-diff-eq-0 linorder-not-less norm-lt-one norm-p.rep-eq power2-eq-square real-sqrt-abs2 real-sqrt-one*)

have *inner*: *inner* (*to-complex ?a*) (*to-complex ?b*) = $a * b *$ *cos gamma*
proof−
have *gamma* = *Mobius-pre-gyrovector-space.angle* (*C t*) (*A t*) (*B t*)
using *assms Mobius-pre-gyrovector-space.get-gamma-def*
by *simp*
then have ∗: *gamma* = *arccos* (*inner* (*Mobius-pre-gyrovector-space.unit* (⊖ (*C t*) ⊕ *A t*)) (*Mobius-pre-gyrovector-space.unit* (⊖ (*C t*) ⊕ *B t*)))
unfolding *Mobius-pre-gyrovector-space.angle-def*

by *simp*
then have *cos gamma* = *inner* (*Mobius-pre-gyrovector-space.unit* (⊖ (*C t*) ⊕ *A t*)) (*Mobius-pre-gyrovector-space.unit* (⊖ (*C t*) ⊕ *B t*))
using *Mobius-pre-gyrovector-space.norm-inner-unit cos-arccos-abs*
by (*metis real-norm-def*)
then have ∗∗: *cos gamma* = (*inner* (*Mobius-pre-gyrovector-space.unit ?a*) (*Mobius-pre-gyrovector-space.unit ?b*))
using *assms*
unfolding *Mobius-pre-gyrovector-space.get-a-def Mobius-pre-gyrovector-space.get-b-def*
by (*simp add*: *inner-commute*)

have *cos*(*gamma*) ∗ *a* ∗ *b* = *inner* (*to-complex ?a*) (*to-complex* (*?b*))
using ∗∗ ‹*a* ≠ *0*› ‹*b* ≠ *0*› *assms*
unfolding *Mobius-pre-gyrovector-space.unit-def*
by (*metis* (*no-types*, *opaque-lifting*) *divide-inverse-commute inner-commute inner-scaleR-right mult.commute nonzero-mult-div-cancel-left times-divide-eq-right*)
then show *?thesis*
by (*simp add*: *field-simps*)
qed

have $(\gamma_m\ (\ominus_m\ ?a \oplus_m\ ?b))^2 = (\gamma_m\ ?a)^2 * (\gamma_m\ ?b)^2 * (1 - 2 * (inner\ (to\text{-}complex\ ?a)\ (to\text{-}complex\ ?b)) + (\langle\langle ?a \rangle\rangle)^2 * (\langle\langle ?b \rangle\rangle)^2)$
using *inner-p.rep-eq m-gamma-equation* **by** *presburger*
also have ... $= (\gamma_m\ ?a)^2 * (\gamma_m\ ?b)^2 * (1 - 2 * a * b * cos(gamma) + (\langle\langle ?a \rangle\rangle)^2 * (\langle\langle ?b \rangle\rangle)^2)$
using *inner* **by** *simp*
finally have $(\gamma_m\ (\ominus_m\ ?a \oplus_m\ ?b))^2\ /\ ((\gamma_m\ ?a)^2 * (\gamma_m\ ?b)^2) = 1 - 2 * a * b * cos(gamma) + (a^2 * b^2)$
using *gammma-factor-m-def gammma-factor-p-def assms* **by** *auto*

moreover

have $(\gamma_m\ (\ominus_m\ ?a \oplus_m\ ?b))^2\ /\ ((\gamma_m\ ?a)^2 * (\gamma_m\ ?b)^2) = ((1 - a^2) * (1 - b^2))\ /\ (1 - c^2)$
proof−
have $(\gamma_m\ ?a)^2\ = 1\ /\ (1 - a^2)$ $(\gamma_m\ ?b)^2\ = 1\ /\ (1 - b^2)$ $(\gamma_m\ ?c)^2\ = 1\ /\ (1 - c^2)$
using *assms*
by (*metis gamma-factor-p-square-norm gammma-factor-m-def gammma-factor-p-def*)+
then show *?thesis*
using *abc*
by *simp*
qed

ultimately
have $1 - 2 * a * b * cos(gamma) + (a^2 * b^2) = ((1 - a^2) * (1 - b^2))\ /\ (1 - c^2)$
by *simp*
then show *?thesis*

using ‹$a \neq 0$› ‹$b \neq 0$› ‹$1 - c^2 \neq 0$›
unfolding *power2-eq-square*
by (*simp add*: *field-simps*)
qed

lemma *T8-25-help3*:
fixes t :: *PoincareDisc otriangle*
assumes $(A\ t) \neq (B\ t)$ $(A\ t) \neq (C\ t)$ $(C\ t) \neq (B\ t)$
$a = \langle\langle$*Mobius-pre-gyrovector-space.get-a t*$\rangle\rangle$ $b = \langle\langle$*Mobius-pre-gyrovector-space.get-b t*$\rangle\rangle$ $c = \langle\langle$*Mobius-pre-gyrovector-space.get-c t*$\rangle\rangle$
gamma = *Mobius-pre-gyrovector-space.get-gamma t*
beta-a $= 1$ / *sqrt* $(1 + a^2)$ *beta-b* $= 1$ / *sqrt* $(1+b^2)$
shows $2 *$ *beta-a*^{2} $* a *$ *beta-b*^{2} $* b *$ *cos gamma* $= (a^2 + b^2 - c^2 - (a*b*c)^2) / ((1 + a^2) * (1 + b^2) * (1-c^2))$
proof−
have $\ominus_m (C\ t) \oplus_m A\ t \neq 0_m$
using *assms*
by (*simp add*: *Mobius-gyrogroup.gyro-equation-right*)
then have $b \neq 0$
using *assms*
unfolding *Mobius-pre-gyrovector-space.get-b-def*
using *gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
by (*simp add*: *Mobius-gyrocarrier'.gyronorm-def*)

have $\ominus_m (C\ t) \oplus_m B\ t \neq 0_m$
using *assms*
by (*simp add*: *Mobius-gyrogroup.gyro-equation-right*)
then have $a \neq 0$
using *assms*
unfolding *Mobius-pre-gyrovector-space.get-a-def*
using *gyroinv-PoincareDisc-def gyroplus-PoincareDisc-def*
by (*simp add*: *Mobius-gyrocarrier'.gyronorm-def*)

have $1 - c^2 \neq 0$
using *assms*
by (*metis abs-norm-cancel dual-order.refl eq-iff-diff-eq-0 linorder-not-less norm-lt-one norm-p.rep-eq power2-eq-square real-sqrt-abs2 real-sqrt-one*)

have *cos gamma* $= (a^2 + b^2 - c^2 - (a*b*c)^2) / (2 * a * b * (1 - c^2))$
using *T8-25-help2 assms*
by *auto*
then have $2 * a * b *$ *cos gamma* $= (a^2 + b^2 - c^2 - (a*b*c)^2) / (1 - c^2)$
using ‹$a \neq 0$› ‹$b \neq 0$›
by *simp*
moreover have (*beta-a*^{2}) $*$ (*beta-b*^{2}) $= 1 / ((1 + a^2) * (1 + b^2))$
using *assms*
by (*simp-all add*: *power2-eq-square*)
ultimately have $2 *$ *beta-a*^{2} $* a *$ *beta-b*^{2} $* b *$ *cos gamma* $= ((a^2 + b^2 - c^2 - (a*b*c)^2) / (1 - c^2)) * 1 / ((1 + a^2) * (1 + b^2))$

```
    by (simp add: field-simps)
  then show ?thesis
    using ‹a ≠ 0› ‹b ≠ 0› ‹1 − c² ≠ 0›
    by simp
 qed


lemma T8-25-help4:
  fixes t :: PoincareDisc otriangle
  assumes (A t) ≠ (B t) (A t) ≠ (C t) (C t) ≠ (B t)
       a = ⟪Mobius-pre-gyrovector-space.get-a t⟫ b = ⟪Mobius-pre-gyrovector-space.get-b
t⟫ c = ⟪Mobius-pre-gyrovector-space.get-c t⟫
          gamma = Mobius-pre-gyrovector-space.get-gamma t
          beta-a = 1 / sqrt (1 + a²) beta-b = 1 / sqrt (1+b²)
    shows 1 − 2 ∗ beta-a² ∗ a ∗ beta-b² ∗ b ∗ cos gamma =
          (1 + (a∗b)² − (a∗c)² − (b∗c)²) / ((1 + a²) ∗ (1 + b²) ∗ (1−c²))
proof−
  have 1 + a² ≠ 0 1 + b² ≠ 0
    by (metis power-one sum-power2-eq-zero-iff zero-neq-one)+

  have 1 − c² ≠ 0
    using assms
   by (metis eq-iff-diff-eq-0 norm-geq-zero norm-lt-one order-less-irrefl power2-eq-square
real-sqrt-abs2 real-sqrt-mult-self real-sqrt-one real-sqrt-pow2)

  have 1 − 2 ∗ beta-a² ∗ a ∗ beta-b² ∗ b ∗ cos gamma = 1 − (a² + b² − c² −
(a∗b∗c)²) / ((1 + a²) ∗ (1 + b²) ∗ (1−c²)) (is ?lhs = 1 − ?nom / ?den)
    using T8-25-help3 assms
    by simp
  also have ... = (?den − ?nom) / ?den
  proof−
    have ?den ≠ 0
      using ‹1 + a² ≠ 0› ‹1 + b² ≠ 0› ‹1 − c² ≠ 0›
      by simp
    then show ?thesis
      by (simp add: field-simps)
  qed
  finally show ?thesis
    by (simp add: field-simps)
qed

lemma T25-help5:
  fixes t :: PoincareDisc otriangle
  assumes (A t) ≠ (B t) (A t) ≠ (C t) (C t) ≠ (B t)
       a = ⟪Mobius-pre-gyrovector-space.get-a t⟫ b = ⟪Mobius-pre-gyrovector-space.get-b
t⟫ c = ⟪Mobius-pre-gyrovector-space.get-c t⟫
          gamma = Mobius-pre-gyrovector-space.get-gamma t
          beta-a = 1 / sqrt (1 + a²) beta-b = 1 / sqrt (1+b²)
    shows (2 ∗ beta-a² ∗ a ∗ beta-b² ∗ b ∗ cos gamma)  / (1 − 2 ∗ beta-a² ∗ a ∗
```

beta-b^2 ∗ *b* ∗ *cos gamma*) =
to-complex ((*of-complex* (a^2)) $\oplus_m$ (*of-complex* (b^2)) $\oplus_m$ ($\ominus_m$ (*of-complex* (c^2)))) (**is** *?lhs* = *?rhs*)
proof −
let *?den* = (*1*+a^2)∗(*1*+b^2)∗(*1*−c^2)
have ∗:*?den* ≠ *0*
using *assms*
by (*smt* (*verit*, *ccfv-threshold*) *divisors-zero norm-geq-zero norm-lt-one not-sum-power2-lt-zero pos2 power-less-one-iff*)

let *?nom1* = a^2 + b^2 − c^2 − ($a{*}b{*}c$)^2 **and** *?nom2* = *1* + ($a{*}b$)^2 − ($a{*}c$)^2 − ($b{*}c$)^2

have *?rhs* = *?nom1* / *?nom2*
using *T8-25-help1*[*OF assms*(*1*−*3*), *of* a^2 b^2 c^2] *assms*
by (*simp add*: *power-mult-distrib*)
also have ... = (*?nom1* / *?den*) / (*?nom2* / *?den*)
using ∗
by *simp*
also have ... = (*2* ∗ *beta-a*^{2} ∗ *a* ∗ *beta-b*^{2} ∗ *b* ∗ *cos gamma*) / (*1* − *2* ∗ *beta-a*^{2} ∗ *a* ∗ *beta-b*^{2} ∗ *b* ∗ *cos gamma*)
using *T8-25-help3*[*OF assms*] *T8-25-help4*[*OF assms*]
by *presburger*
finally show *?thesis*
by (*simp add*: *cos-of-real*)
qed

lemma *T25-MobiusCosineLaw*:
fixes *t* :: *PoincareDisc otriangle*
assumes (*A t*) ≠ (*B t*) (*A t*) ≠ (*C t*) (*C t*) ≠ (*B t*)
a = ⟪*Mobius-pre-gyrovector-space.get-a t*⟫ *b* = ⟪*Mobius-pre-gyrovector-space.get-b t*⟫ *c* = ⟪*Mobius-pre-gyrovector-space.get-c t*⟫
gamma = *Mobius-pre-gyrovector-space.get-gamma t*
beta-a = *1* / *sqrt* (*1* + a^2) *beta-b* = *1* / *sqrt* (*1*+b^2)
shows c^2 = *to-complex* ((*of-complex* (a^2)) $\oplus_m$ (*of-complex* (b^2)) $\oplus_m$ ($\ominus_m$ (*of-complex*
(*2* ∗ *beta-a*^{2} ∗ *a* ∗ *beta-b*^{2} ∗ *b* ∗ *cos*(*gamma*) /
(*1* − *2* ∗ *beta-a*^{2} ∗ *a* ∗ *beta-b*^{2} ∗ *b* ∗ *cos gamma*)))))
proof −
let *?a* = *of-complex* (a^2) **and** *?b* = *of-complex* (b^2) **and** *?c* = *of-complex* (c^2)
have *norm* (c^2) < *1*
using *assms*
by (*simp add*: *norm-geq-zero norm-lt-one power-less-one-iff*)+
then have c^2 = *to-complex* (*?a* $\oplus_m$ *?b* $\oplus_m$ ($\ominus_m$ (*?a* $\oplus_m$ *?b* $\oplus_m$ ($\ominus_m$ *?c*))))
using *Mobius-gyrocommutative-gyrogroup.gyroautomorphic-inverse Mobius-gyrogroup.gyrominus-def Mobius-gyrogroup.gyro-inv-idem Mobius-gyrogroup.oplus-ominus-cancel*
by (*metis* (*mono-tags*, *lifting*) *Mobius-gyrocarrier'.to-carrier mem-Collect-eq norm-of-real real-norm-def*)

then show *?thesis*
using *T25-help5 assms*
by *auto*
qed

abbreviation *add-complex* (**infixl** $\oplus_{mc}$ *100*) **where**
add-complex c1 c2 ≡ *to-complex* (*of-complex c1* $\oplus_m$ *of-complex c2*)

lemma *T-MobiusPythagorean*:
fixes *t* :: *PoincareDisc otriangle*
assumes (*A t*) ≠ (*B t*) (*A t*) ≠ (*C t*) (*C t*) ≠ (*B t*)
a = ⟪*Mobius-pre-gyrovector-space.get-a t*⟫ *b* = ⟪*Mobius-pre-gyrovector-space.get-b t*⟫ *c* = ⟪*Mobius-pre-gyrovector-space.get-c t*⟫
gamma = *Mobius-pre-gyrovector-space.get-gamma t gamma* = *pi* / *2*
shows $c^2 = a^2 \oplus_{mc} b^2$
using *assms T25-MobiusCosineLaw*[*OF assms*(*1*−*7*)]
by (*metis* (*no-types*, *opaque-lifting*) *Mobius-gyrogroup.oplus-ominus-cancel cos-of-real-pi-half diff-self div-0 m-gamma-h1 mult.commute mult-zero-left of-real-divide of-real-numeral*)

end
theory *Poincare*
imports *Complex-Main HOL−Analysis.Inner-Product GammaFactor*
begin

typedef *PoincareDisc* = {*z*::*complex*. *cmod z* < *1*}
by (*rule exI* [**where** *x*=*0*], *auto*)

setup-lifting *type-definition-PoincareDisc*

abbreviation *to-complex* :: *PoincareDisc* ⇒ *complex* **where**
to-complex ≡ *Rep-PoincareDisc*
abbreviation *of-complex* :: *complex* ⇒ *PoincareDisc* **where**
of-complex ≡ *Abs-PoincareDisc*

lemma *poincare-disc-two-elems*:
shows ∃ *z1 z2*::*PoincareDisc*. *z1* ≠ *z2*
proof−
have *cmod 0* < *1*
by *simp*
moreover
have *cmod* (*1*/*2*) < *1*
by *simp*
moreover
have (*0*::*complex*) ≠ *1*/*2*
by *simp*
ultimately
show *?thesis*
by *transfer blast*
qed

lift-definition *inner-p* :: *PoincareDisc* ⇒ *PoincareDisc* ⇒ *real* (**infixl** · *100*) **is** *inner* **.**

lift-definition *norm-p* :: *PoincareDisc* ⇒ *real* (⟪-⟫ [*100*] *101*) **is** *norm* **.**

lemma *norm-lt-one*:
shows ⟪*u*⟫ < *1*
by *transfer simp*

lemma *norm-geq-zero*:
shows ⟪*u*⟫ ≥ *0*
by *transfer simp*

lemma *square-norm-inner*:
shows $(\langle\langle u\rangle\rangle)^2 = u \cdot u$
by *transfer* (*simp add*: *dot-square-norm*)

lift-definition *gammma-factor-p* :: *PoincareDisc* ⇒ *real* (γ_p) **is** *gamma-factor* **.**

lemma *gamma-factor-p-nonzero* [*simp*]:
shows γ_p *u* ≠ *0*
apply *transfer*
unfolding *gamma-factor-def*
using *gamma-factor-nonzero*
by *auto*

lemma *gamma-factor-p-positive* [*simp*]:
shows γ_p *u* > *0*
by *transfer* (*simp add*: *gamma-factor-positive*)

lemma *norm-square-gamma-factor-p*:
shows (⟪*u*⟫)^*2* = *1* − *1* / (γ_p *u*)^*2*
by *transfer* (*simp add*: *norm-square-gamma-factor*)

lemma *norm-square-gamma-factor-p'*:
shows (⟪*u*⟫)^*2* = ((γ_p *u*)^*2* − *1*) / (γ_p *u*)^*2*
by *transfer* (*simp add*: *norm-square-gamma-factor'*)

lemma *gamma-factor-p-square-norm*:
shows $(\gamma_p\ u)^2 = 1\ /\ (1 - (\langle\langle u\rangle\rangle)^2)$
by *transfer* (*simp add*: *gamma-factor-square-norm*)

end

References

[1] A. A. Ungar. *Analytic Hyperbolic Geometry: Mathematical Foundations and Applications*. World Scientific, Singapore, 2005.